SIMILARITY SEARCH
The Metric Space Approach

Pavel Zezula, Giuseppe Amato,
Vlastislav Dohnal, Michal Batko

Table of Contents

Foundations of metric space searching

1. **distance searching problem in metric spaces**
2. metric distance measures
3. similarity queries
4. basic partitioning principles
5. principles of similarity query execution
6. policies to avoid distance computations
7. metric space transformations
8. principles of approximate similarity search
9. advanced issues

Distance searching problem

- Search problem:
  - Data type
  - The method of comparison
  - Query formulation
- Extensibility:
  - A single indexing technique applied to many specific search problems quite different in nature

Distance searching problem

- Traditional search:
  - Exact (partial, range) retrieval
  - Sortable domains of data (numbers, strings)
- Perspective search:
  - Proximity
  - Similarity
  - Dissimilarity
  - Distance
  - Not sortable domains (Hamming distance, color histograms)

Distance searching problem

Definition (divide and conquer):

- Let $\mathcal{D}$ be a domain, *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 proximity queries are answered efficiently.

Distance searching problem

- Metric space as similarity search abstraction
  - Distances used for searching
  - No coordinates – no data space partitioning
  - Vector versus metric spaces
- Three reasons for metric indexes:
  - No other possibility
  - Comparable performance for special cases
  - High extensibility

Metric space

- $\mathcal{M} = (\mathcal{D}, d)$
  - 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) does not hold
- If all objects at distance 0 are considered as single objects, 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)

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

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

Foundations of metric space searching

1. distance searching problem in metric spaces
2. **metric distance measures**
3. similarity queries
4. basic partitioning principles
5. principles of similarity query execution
6. policies to avoid distance computations
7. metric space transformations
8. principles of approximate similarity search
9. advanced issues

Distance measures

- Discrete
  - functions which return only a small (predefined) set of values
- Continuous
  - functions in which the cardinality of the set of values returned is very large or infinite.

Minkowski distances

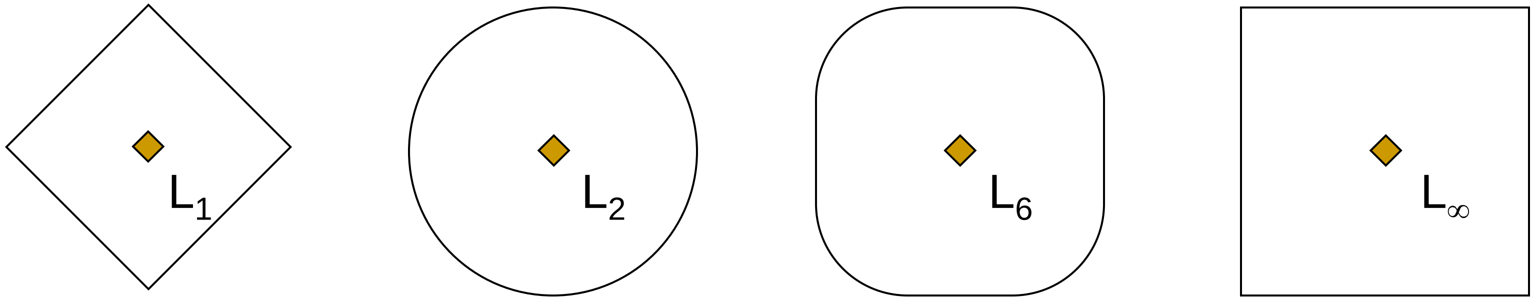
- Also called the L_p metrics
- Defined on *n* dimensional vectors

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

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

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

- *M* – positive semidefinite matrix $n \times n$
  - if $M = diag(w_1, \ldots, w_n) \rightarrow$ 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 red one

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

Example (continue)

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

$$M = \begin{array}{cc} \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.9 \\ 0.0 & 0.9 & 1.0 \end{bmatrix} & \begin{array}{l} \text{blue} \\ \text{red} \\ \text{orange} \end{array} \\ \begin{array}{ccc} \text{blue} & \text{red} & \text{orange} \end{array} & \end{array}$$

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

Edit distance

- Also called the Levenstein distance:
  - 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 *x* with *c*

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

Edit distance - 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}(“combine”,”combination”) = 9
  replacement $e \rightarrow a$, insertion *t,i,o,n*
  d_{edit}(“combination”,” 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

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

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

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

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

Hausdorff distance

- Distance measure for sets
- Compares elements by a distance d_e

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

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.)

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

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

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

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

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

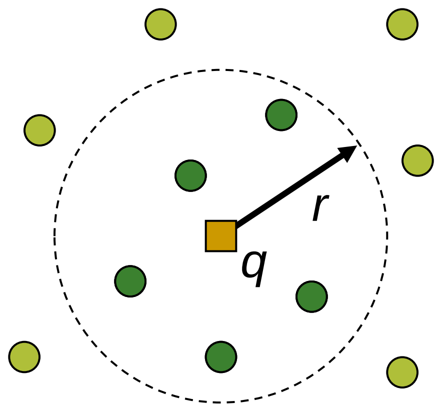
Foundations of metric space searching

1. distance searching problem in metric spaces
2. metric distance measures
3. **similarity queries**
4. basic partitioning principles
5. principles of similarity query execution
6. policies to avoid distance computations
7. metric space transformations
8. principles of approximate similarity search
9. advanced issues

Similarity Queries

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

Similarity Range Query

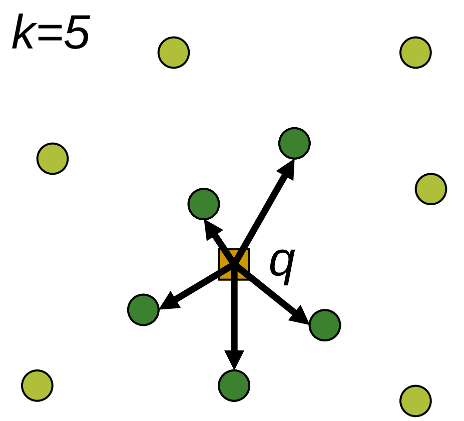


- range query
  - $R(q,r) = \{ x \in X \mid d(q,x) \leq r \}$

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



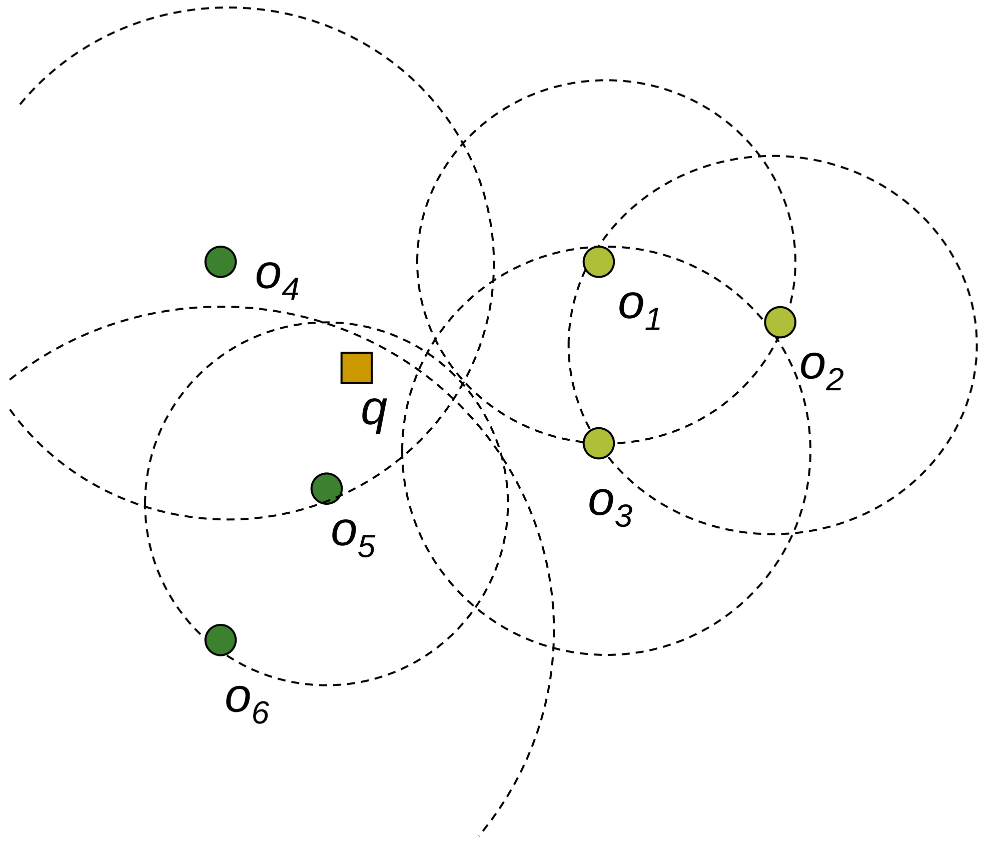
… five closest museums to my hotel …

Reverse Nearest Neighbor

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

… *all hotels with a specific museum as a nearest cultural heritage cite* …

Example of *2-RNN*



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

Similarity Queries

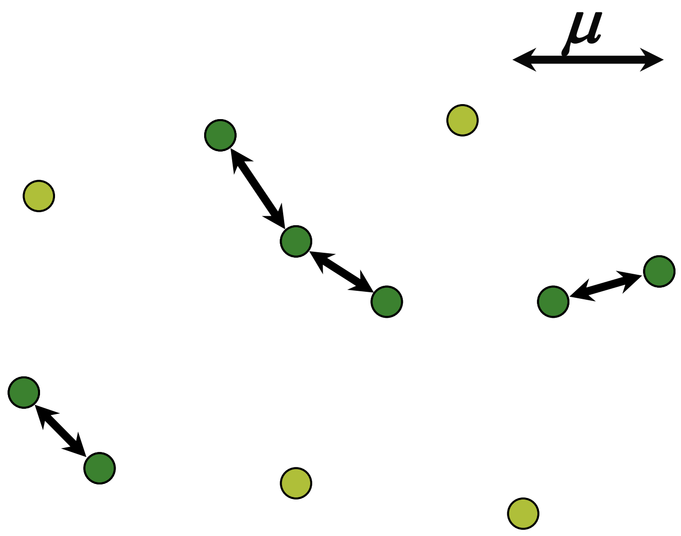
- similarity join of two data sets

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

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

- similarity self join ⇔ *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!!!

The $\mathcal{A}_0$ Algorithm

- For each predicate *i*
  - objects delivered in decreasing similarity
  - incrementally build sets X_i with best matches till

$$\forall i \, |\cap_i X_i| = k$$

- For all $o \in \cup_i X_i$
  - consider all query predicates
  - establish the final rank (fuzzy algebra, weighted sets, etc.)

Foundations of Metric Space Searching

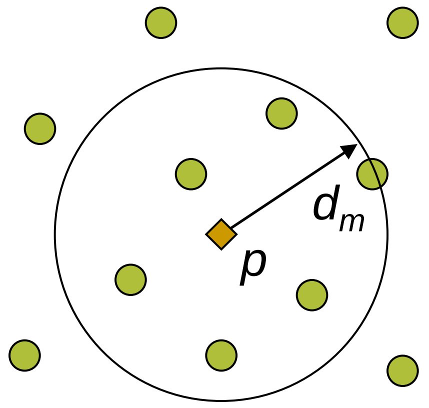
1. distance searching problem in metric spaces
2. metric distance measures
3. similarity queries
4. **basic partitioning principles**
5. principles of similarity query execution
6. policies to avoid distance computations
7. metric space transformations
8. principles of approximate similarity search
9. advanced issues

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

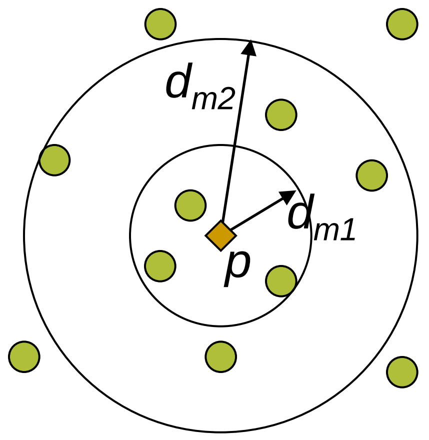
Ball partitioning

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



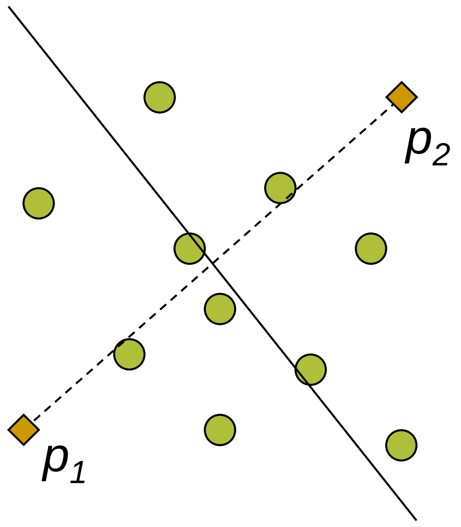
Multi-way ball partitioning

- 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: $\{ x \in X \mid d(p,x) > d_{m2} \}$



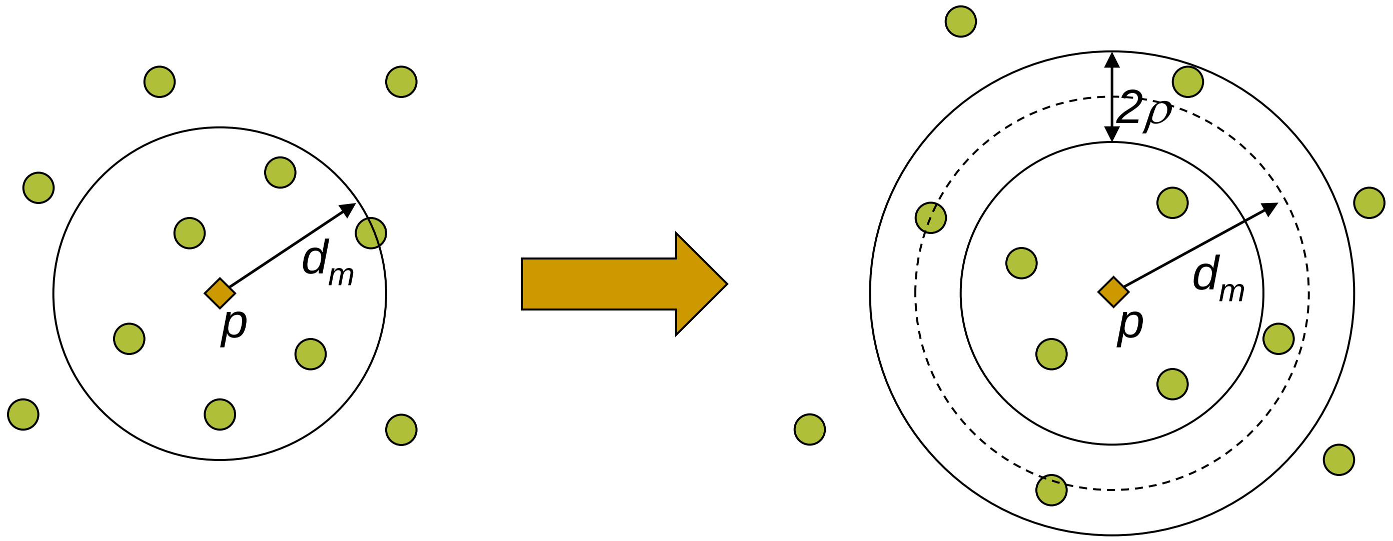
Generalized Hyper-plane Partitioning

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



Excluded Middle Partitioning

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



- Excluded set: otherwise

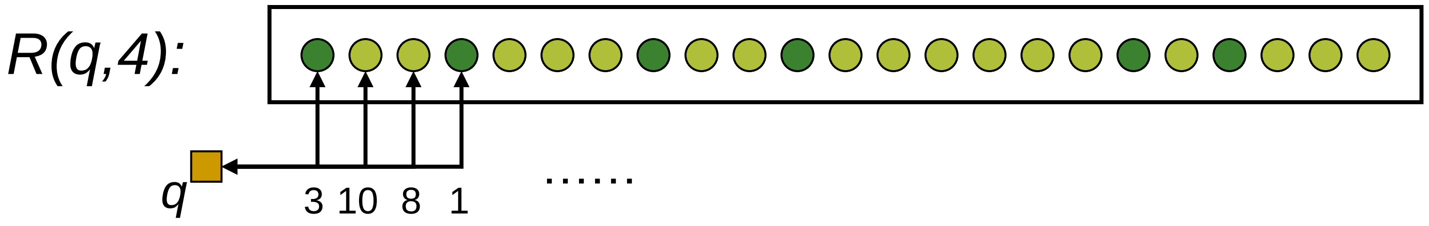
Foundations of metric space searching

1. distance searching problem in metric spaces
2. metric distance measures
3. similarity queries
4. basic partitioning principles
5. **principles of similarity query execution**
6. policies to avoid distance computations
7. metric space transformations
8. principles of approximate similarity search
9. advanced issues

Basic Strategies

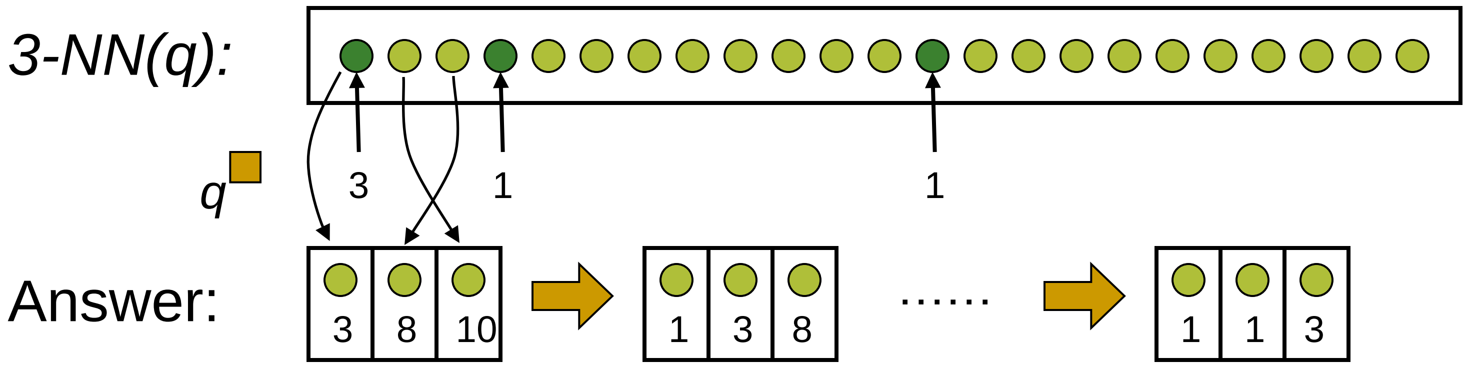
- Costs to answer a query are influenced by
  - Partitioning principle
  - Query execution algorithm

- Sequential organization & range query *R*(*q*,*r*)
  - All database objects are consecutively scanned and *d*(*q*,*o*) are evaluated.
  - Whenever *d*(*q*,*o*) ≤ *r*, *o* is reported on result



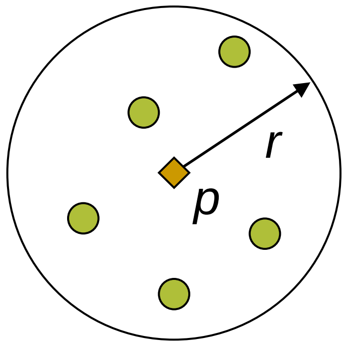
Basic Strategies (cont.)

- 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

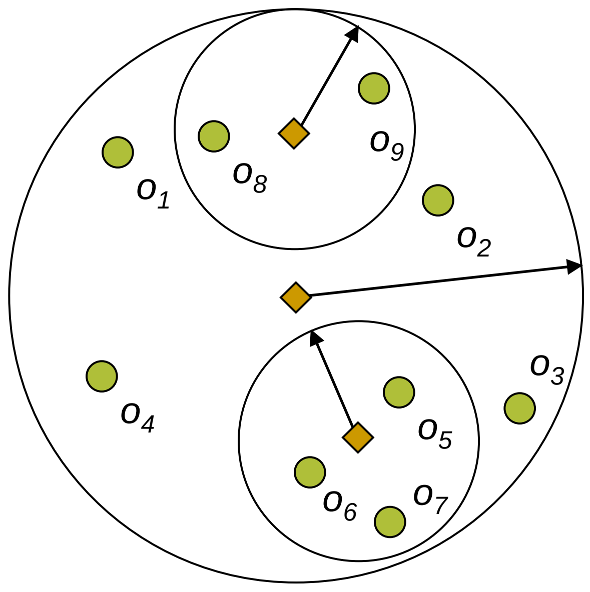
- A hierarchy of entries (nodes) $N=(G,\mathcal{R}(G))$
  - G = {e | e is object or e is another entry}
  - *Bounding region* $\mathcal{R}(G)$ covers all elements of *G.*
  - E.g. *ball region*: $\forall o,\ d(o,p) \leq r$
  - Each element belongs exactly to one *G.*
  - There is one *root* entry *N.*

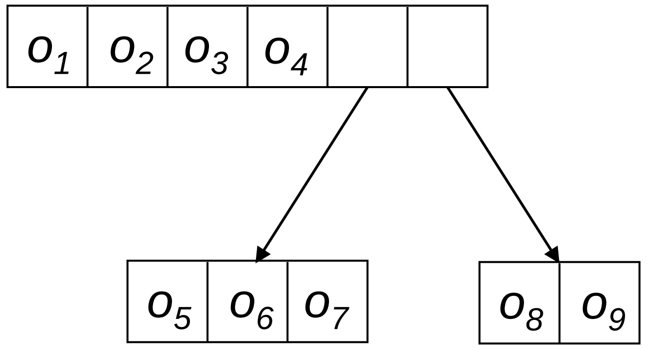


- Any similarity query *Q* returns a set of objects
  - We can define $\mathcal{R}(Q)$ which covers all objects in response.

Example of Index Organization

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





Range Search Algorithm

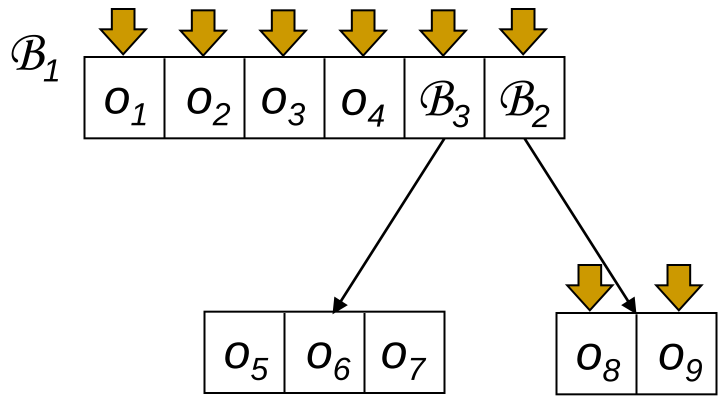
Given $Q=R(q,r)$:

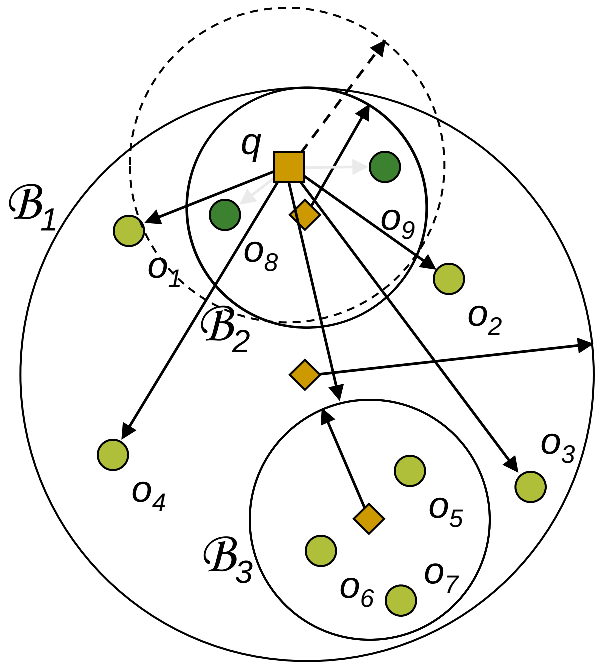
- Start at the root.
- In the current node $N=(G,\mathcal{R}(G))$, process all elements:
  - object element $o_j \in G$:
    - if $d(q,o_j) \leq r$, report o_j on output.
  - non-object element $N'=(G',\mathcal{R}(G'))\in G$
    - if $\mathcal{R}(G')$ and $\mathcal{R}(Q)$ intersect, recursively search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

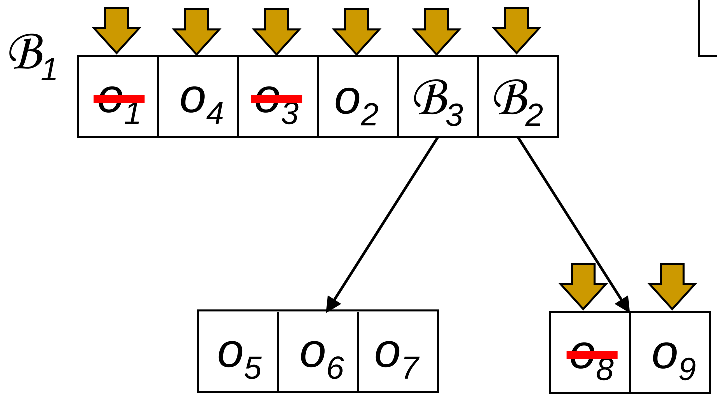
- While *PR* is not empty, repeat:
  - Pick an entry $N=(G,\mathcal{R}(G))$ from *PR.*
  - For each object element $o_j \in G$:
    - if $d(q,o_j) \leq r$, add o_j to the response. Update r and $\mathcal{R}(Q)$.
      - Remove entries from *PR* that cannot intersect the query.
  - For each non-object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

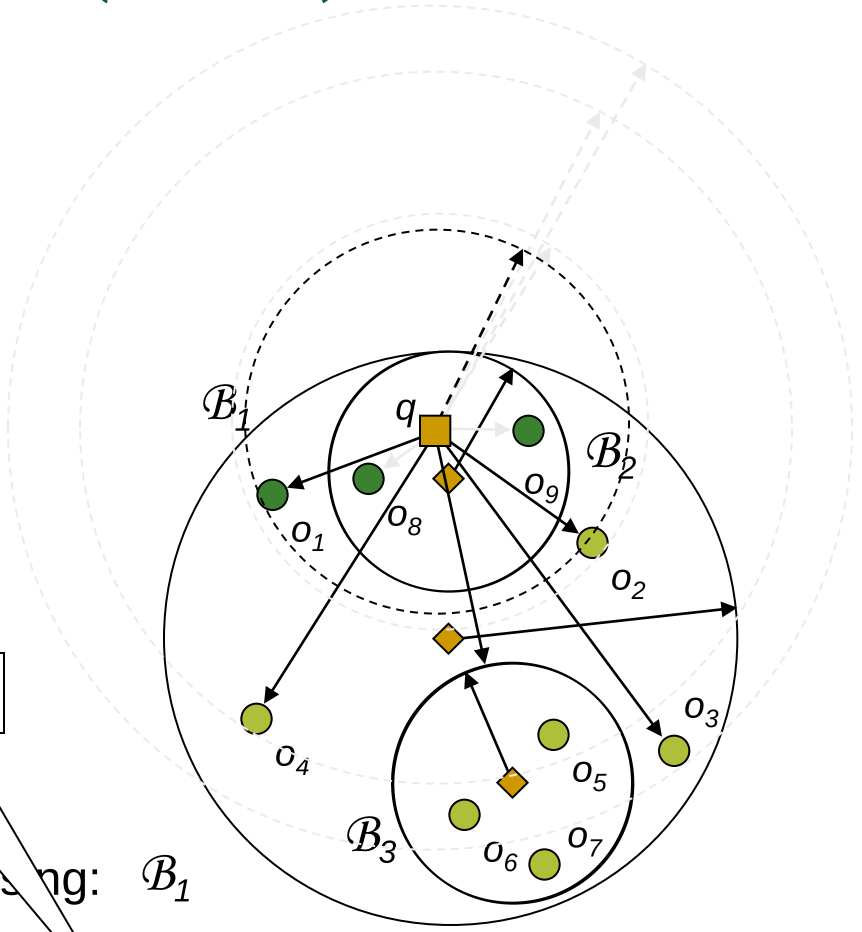


Final result

Processing: $\mathcal{B}_1$

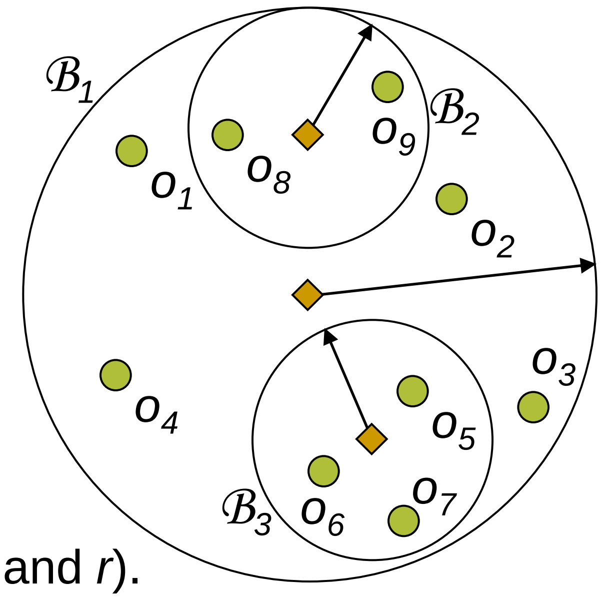
PR= $\mathcal{B}_2$

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

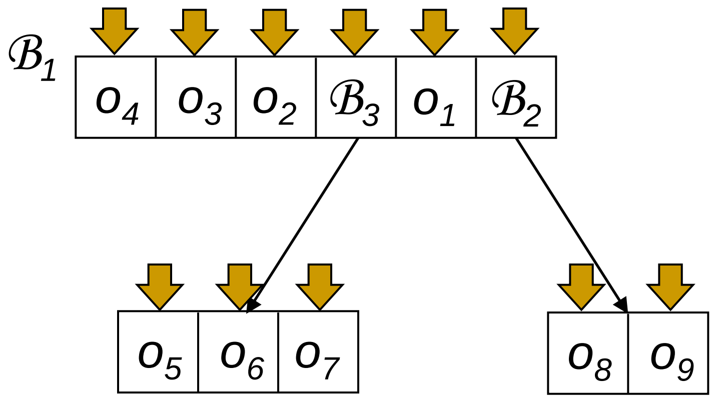
- Based on *priority queue PR* again
  - Each element e_t in *PR* knows also the distance $d_t(q,e_t)$.
  - Entries in the queue are sorted with respect to these distances.
- Initialization:
  - Insert the root element with the distance *0* into *PR*.

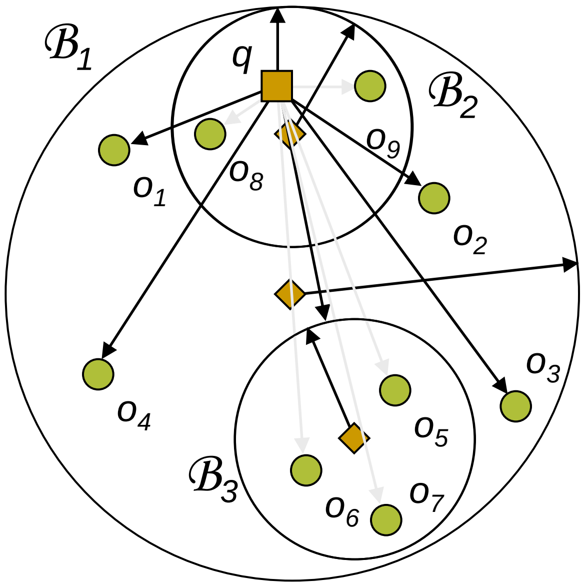
Incremental *NN* Search (cont.)

- 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

Queue = [illegible] $(o_3,7)$ $(o_3,7)$

Response = o_8 , o_9 , o_1 , o_2 , o_5 , o_4 , o_6 , o_3 , o_7

Foundations of metric space searching

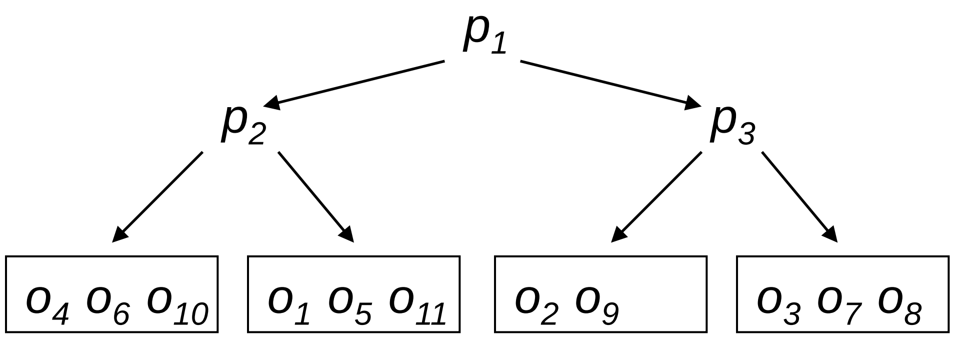
1. distance searching problem in metric spaces
2. metric distance measures
3. similarity queries
4. basic partitioning principles
5. principles of similarity query execution
6. **policies to avoid distance computations**
7. metric space transformations
8. principles of approximate similarity search
9. advanced issues

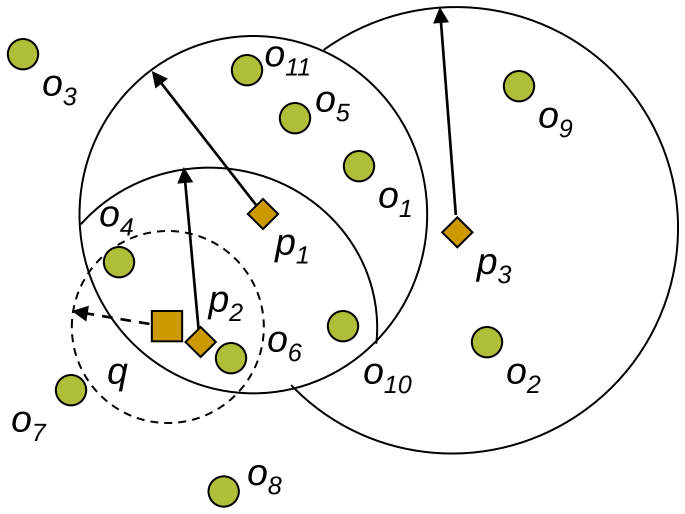
Avoiding Distance Computations

- 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
  - object-pivot
  - range-pivot
  - pivot-pivot
  - double-pivot
  - pivot filtering

Explanatory Example

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

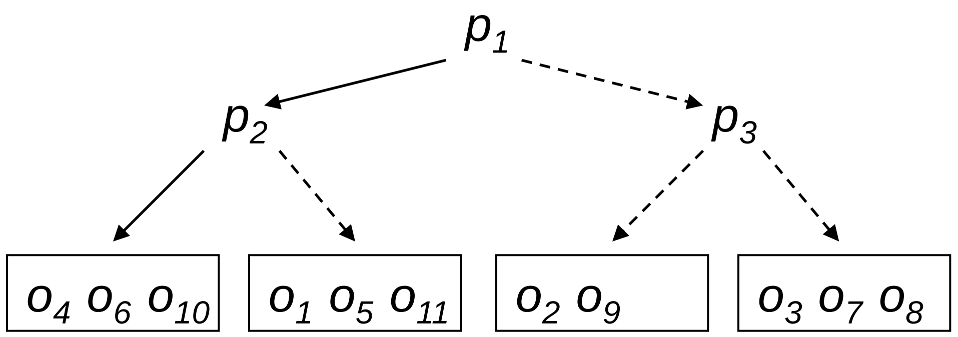




- Range query $R(q,r)$
  - Sequential scan needs 11 distance computations.
  - Reported objects: $\{o_4, o_6\}$

Object-Pivot Distance Constraint

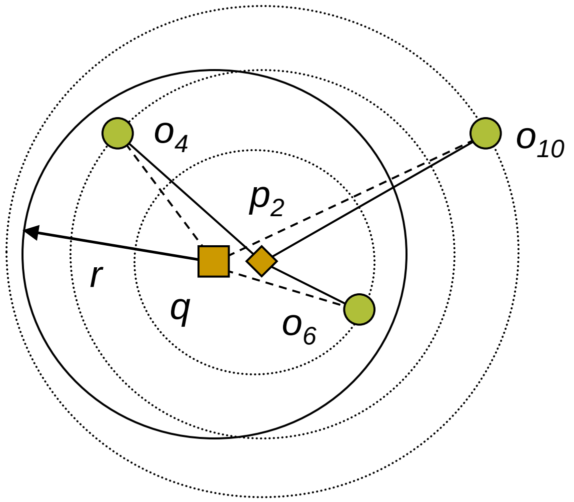
- Usually applied in leaf nodes
- Assume the left-most leaf is visited
  - Distances from q to o_4, o_6, o_{10} must be computed



- 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 dotted circle



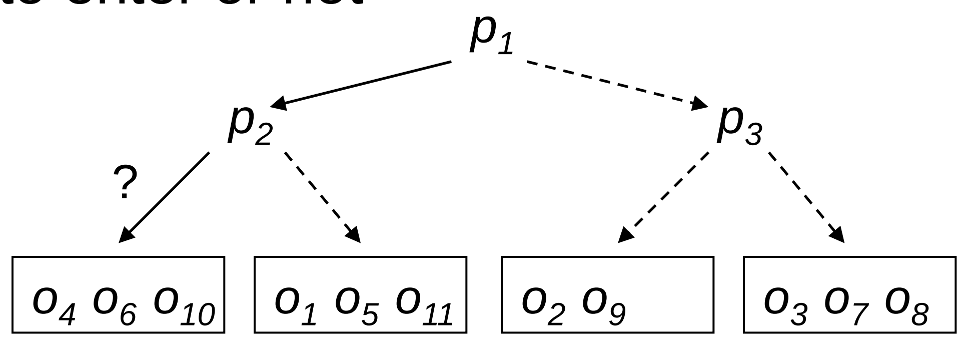
Object-Pivot Constraint (summary)

- 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:

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

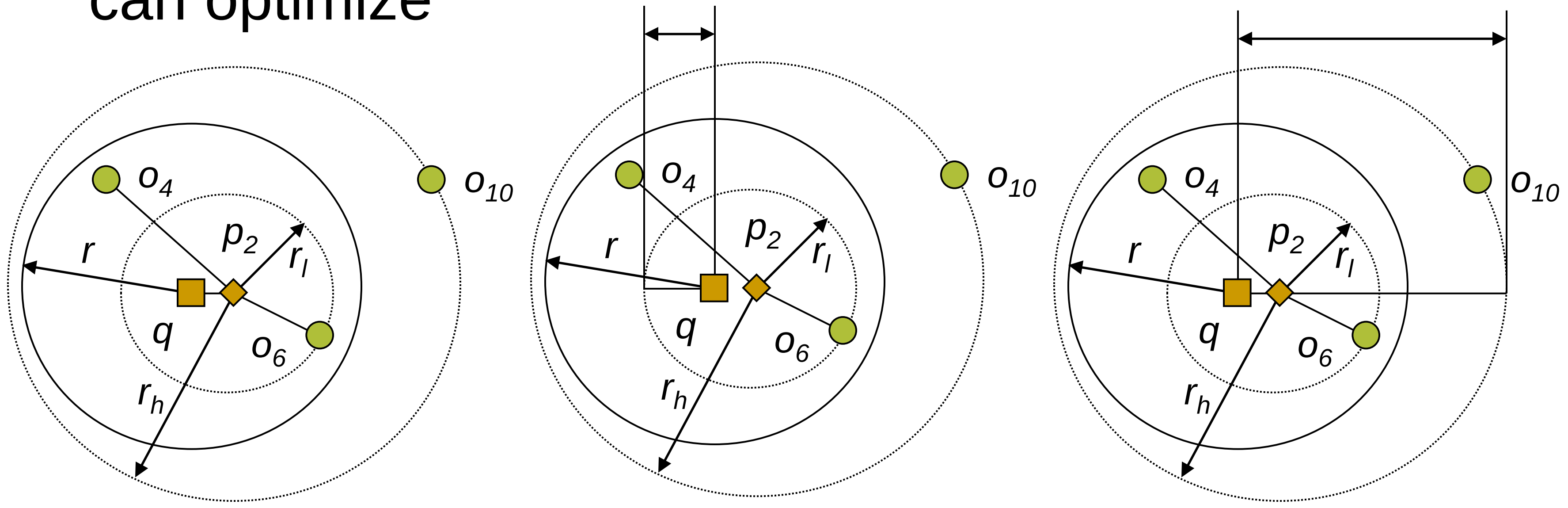
Range-Pivot Distance Constraint

- 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.)

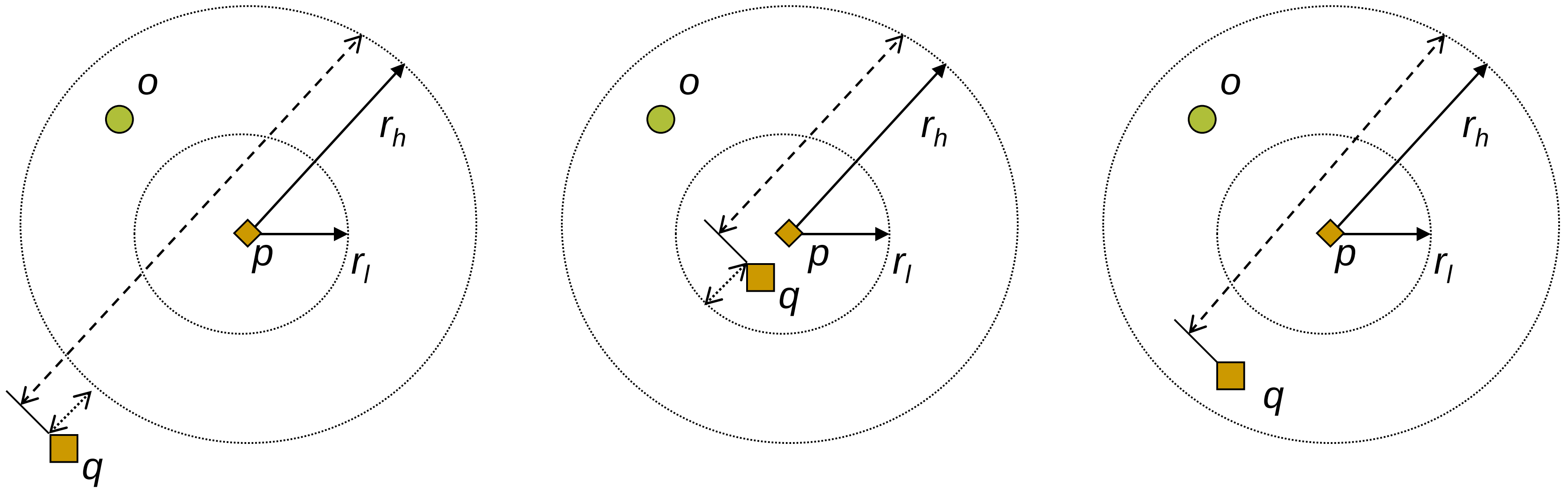
- Knowing interval $[r_l, r_h]$ of distance in the leaf, we can optimize



- 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:



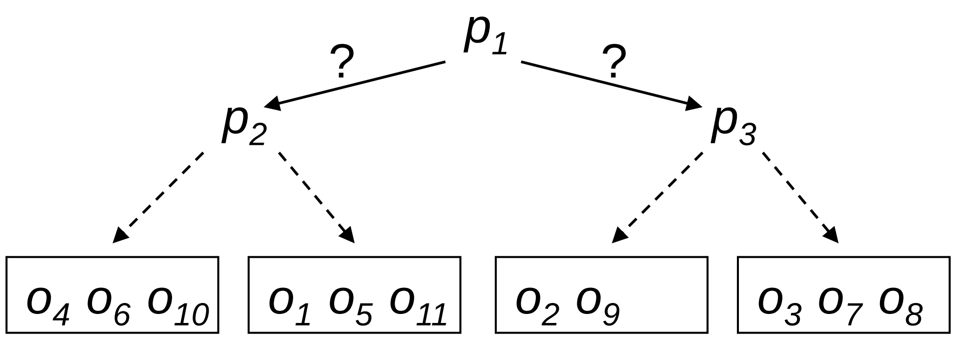
Range-Pivot Constraint (summary)

- 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:

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

Pivot-Pivot Distance Constraint

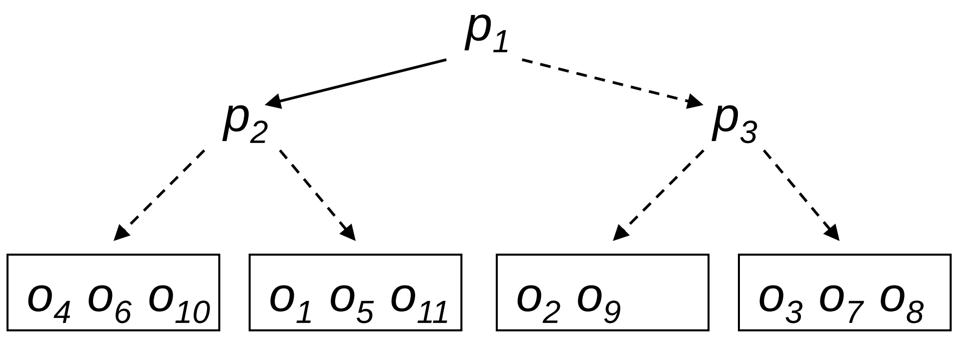
- In internal nodes we can do more
- Assume the root node is examined

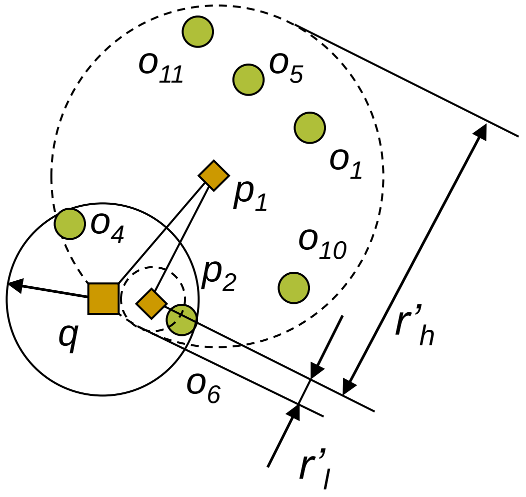


- We can apply the *range-pivot constraint* to decide which sub-trees must be visited
  - The ranges are known since during building phase all data objects were compared with p_1.

Pivot-Pivot Constraint (cont.)

- Suppose we have followed the left branch (to p_2)
- Knowing the distance $d(p_1,p_2)$ and using $d(q,p_1)$
  - we can apply the *object-pivot constraint* $\rightarrow d(q,p_2) \in [r_l', r_h']$

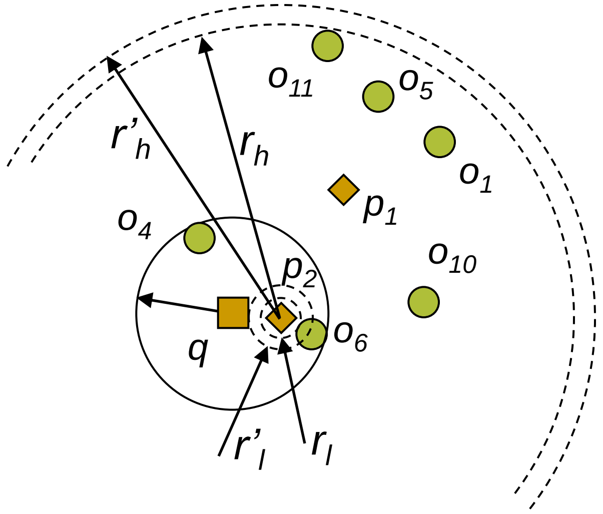




- We also know range of distances in p_2's sub-trees: $d(o,p_2) \in [r_l, r_h]$

Pivot-Pivot Constraint (cont.)

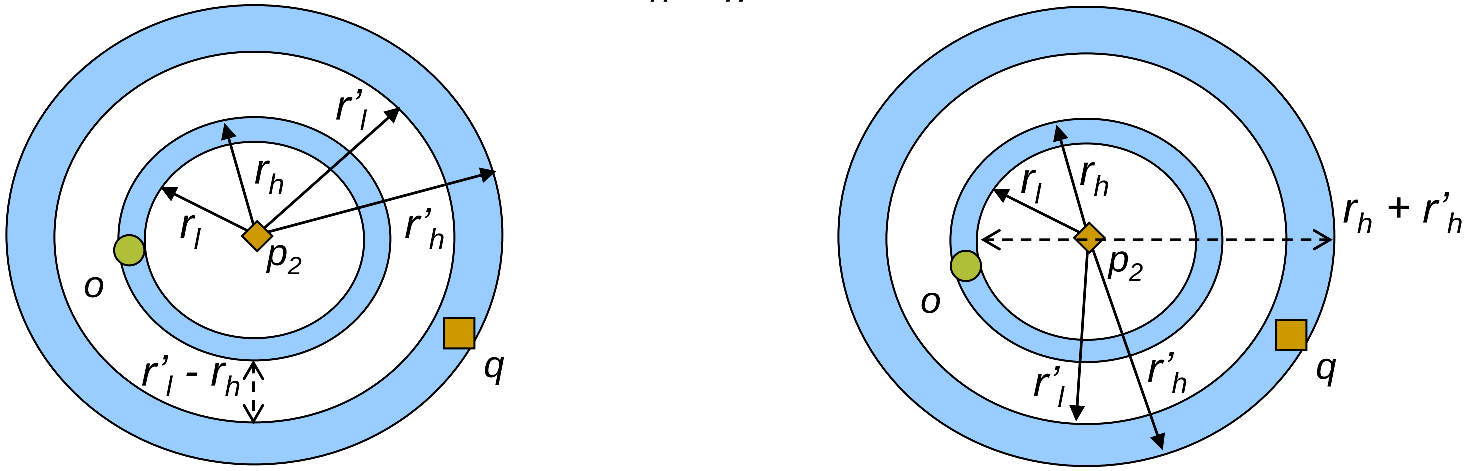
- Having
  - $d(q,p_2) \in [r_l', r_h']$
  - $d(o,p_2) \in [r_l, r_h]$



- Both ranges intersect $\rightarrow$ lower bound on $d(q,o)$ is 0!
- Upper bound is $r_h + r_h'$

Pivot-Pivot Constraint (cont.)

- If ranges do not intersect, there are two possibilities.
- The first is: $[r_l,r_h]$ is less than $[r_l',r_h']$
  - The lower bound (left) is $r_l' - r_h$
  - A view of the upper bound r_h+r_h' (right)



- The second is inverse - the lower limit is $r_l - r_h'$

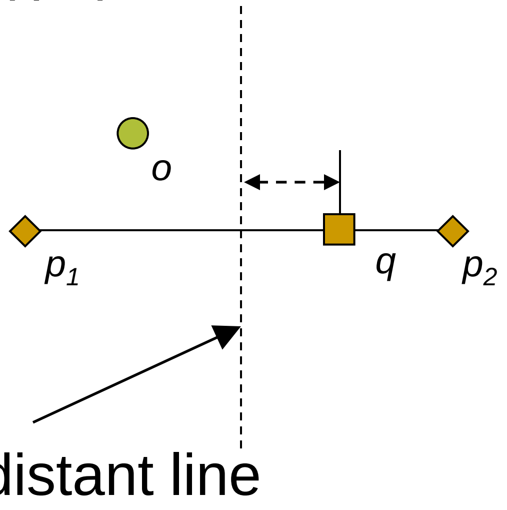
Pivot-Pivot Constraint (summary)

- 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:

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

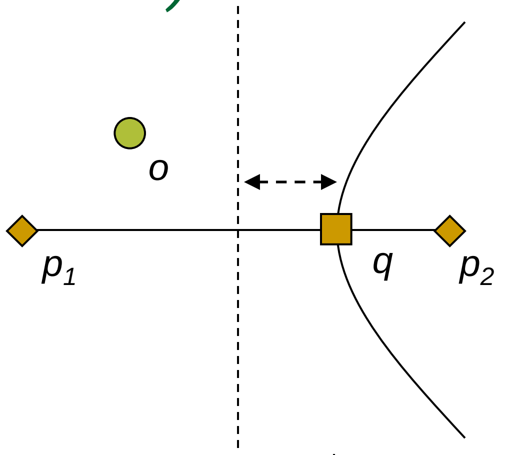
Double-Pivot Distance Constraint

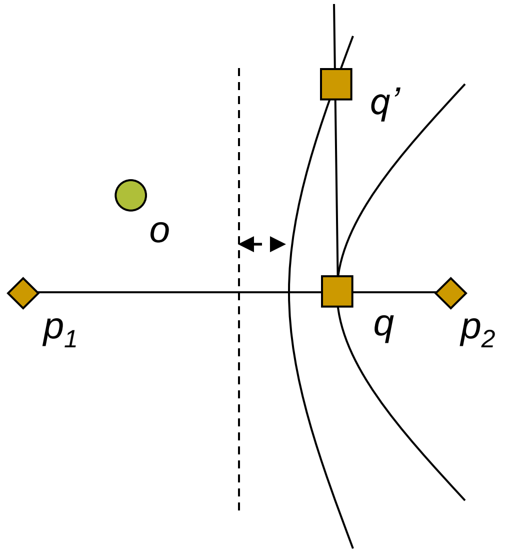
- Previous constraints use just one pivot along with ball partitioning.
- Applying generalized hyper-plane, we have two pivots.
  - No upper bound on $d(q,o)$ can be defined!



Double-Pivot Constraint (cont.)

- If *q* and *o* are in different subspaces
  - Lower bound is $(d(q,p_1) – d(q,p_2))/2$
  - Hyperbola shows the positions with constant lower bound.
- Moving *q* up (so “visual” distance from equidistant line is preserved), decreases the lower bound.
- If *q* and *o* are in the same subspace
  - the lower bound is zero





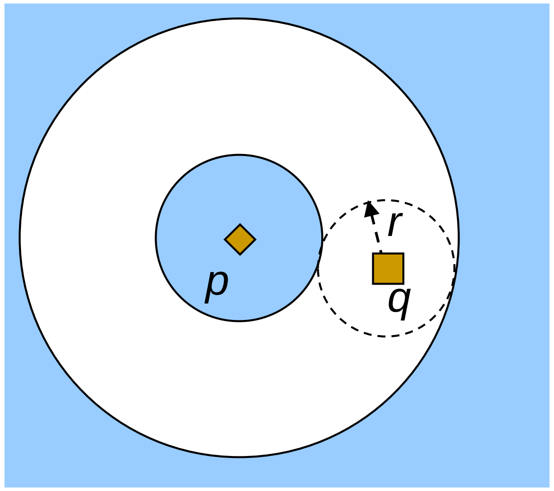
Double-Pivot Constraint (summary)

- 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:

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

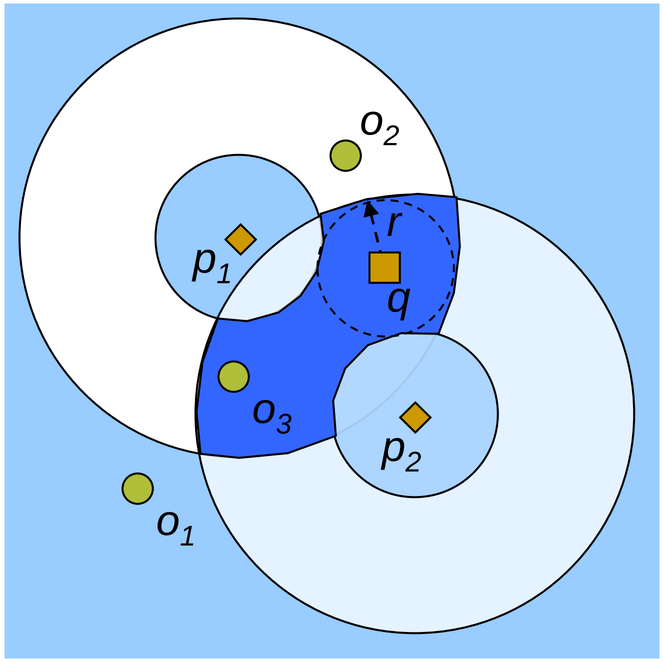
Pivot Filtering

- Extended object-pivot constraint
  - Uses more pivots
- Uses triangle inequality for pruning
- All distances between objects and a pivot p are known
- Prune 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 $P = \{ p_1, p_2, p_3, \ldots, p_n \}$. We define a mapping function Ψ: $(\mathcal{D},d) \rightarrow (\mathbb{R}^n, L_\infty)$ as follows:

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

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

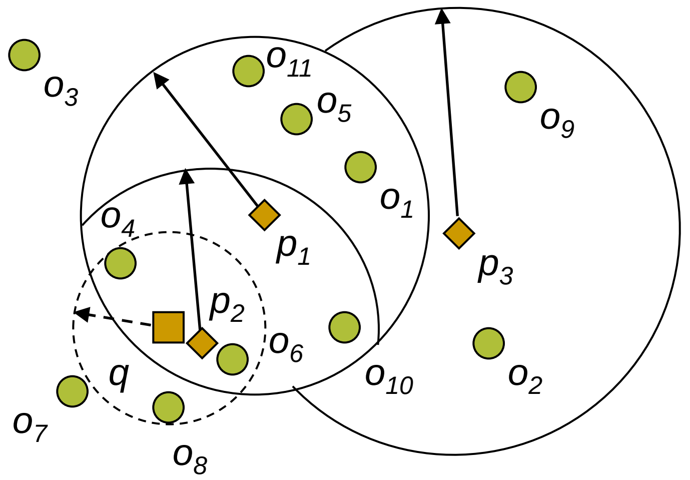
$$L_\infty(\Psi(o), \Psi(q)) \leq d(q,o)$$

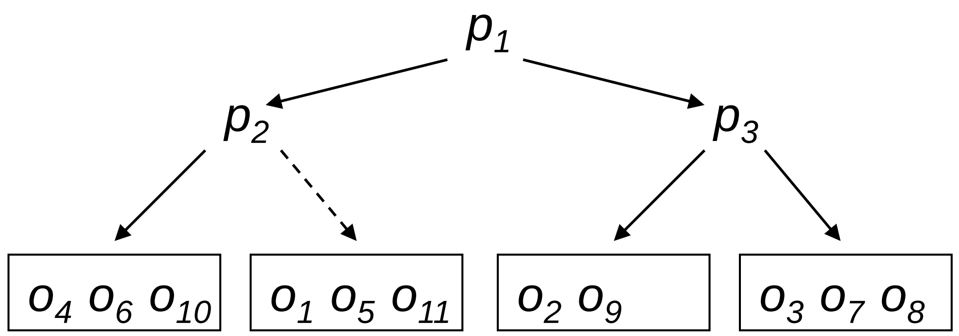
Pivot Filtering (consideration)

- 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 $d()$.

Constraints & Explanatory Example 1

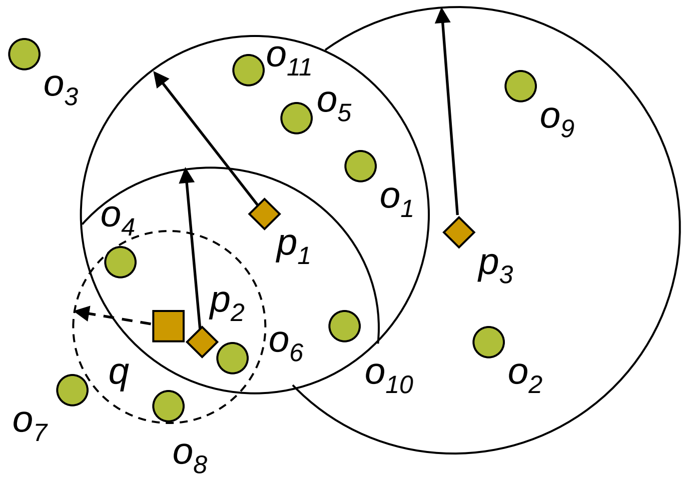
- Range query $R(q,r) = \{o_4,o_6,o_8\}$
  - Sequential scan: 11 distance computations
  - No constraint: 3+8 distance computations

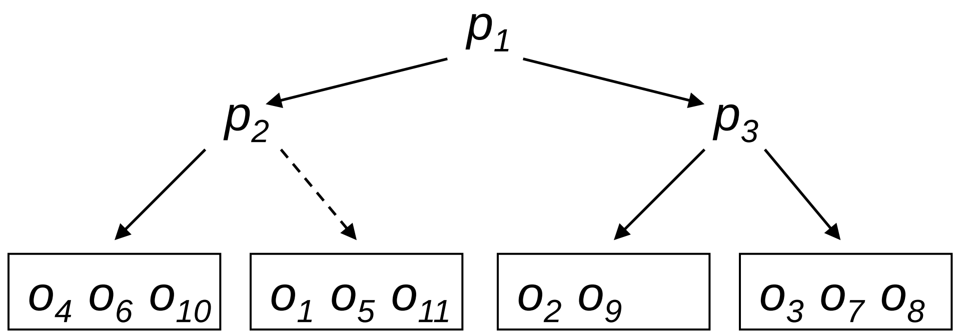




Constraints & Explanatory Example 2

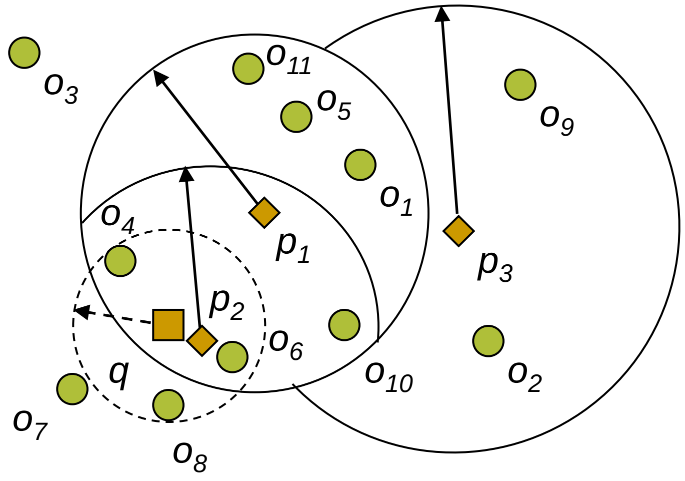
- Range query $R(q,r) = \{o_4,o_6,o_8\}$
  - Only *object-pivot* in leaves: 3+2 distance computations
    - o_6 is included without computing $d(q,o_6)$
    - $o_{10}, o_2, o_9, o_3, o_7$ are eliminated without computing.

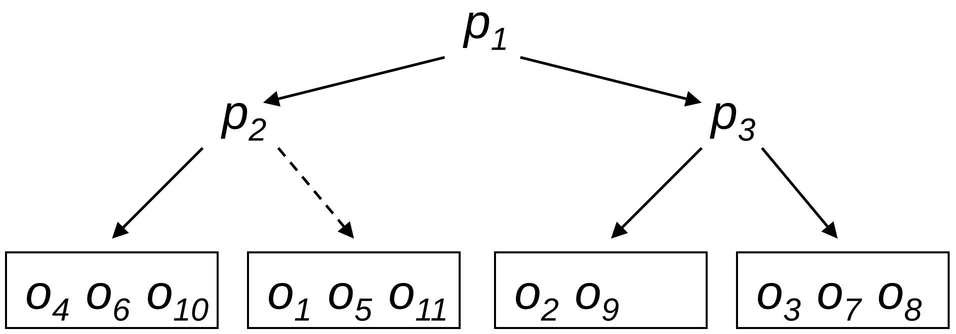




Constraints & Explanatory Example 3

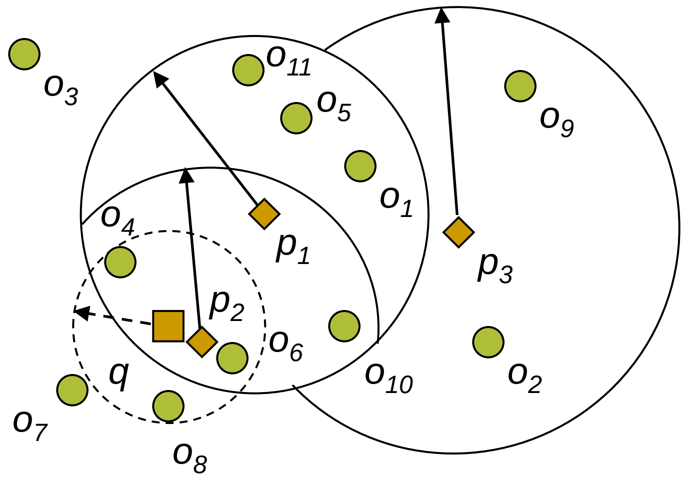
- Range query $R(q,r) = \{o_4,o_6,o_8\}$
  - Only *range-pivot:* 3+6 distance computations
    - o_2 ,o_9 are pruned.
  - Only *range-pivot +pivot-pivot:* 3+6 distance computations

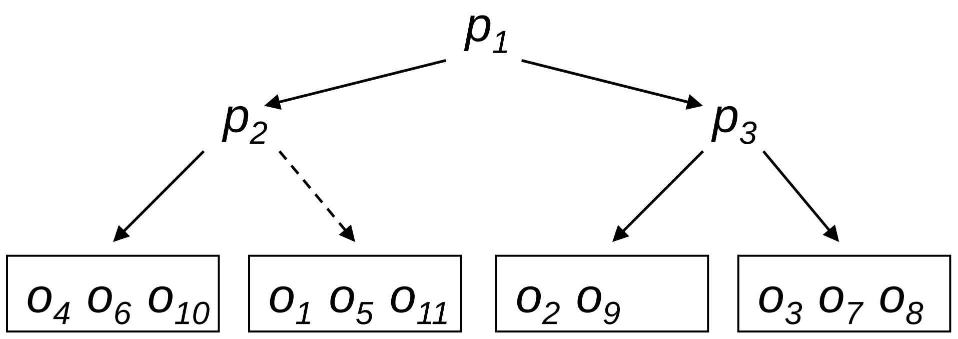




Constraints & Explanatory Example 4

- Range query $R(q,r) = \{o_4,o_6,o_8\}$
  - Assume: objects know distances to pivots along paths to the root.
  - Only *pivot filtering:* 3+3 distance computations (to o_4 , o_6 , o_8)
  - All constraints together: 3+2 distance computations (to o_4 ,o_8)





Foundations of metric space searching

1. distance searching problem in metric spaces
2. metric distance measures
3. similarity queries
4. basic partitioning principles
5. principles of similarity query execution
6. policies to avoid distance computations
7. **metric space transformations**
8. principles of approximate similarity search
9. advanced issues

Metric Space Transformation

- Change one metric space into another
  - Transformation of the original objects
  - Changing the metric function
  - Transforming both the function and the objects
- Metric space embedding
  - Cheaper distance function
- User-defined search functions

Metric Space Transformation

- $\mathcal{M}_1 = (\mathcal{D}_1, d_1) \Longrightarrow \mathcal{M}_2 = (\mathcal{D}_2, d_2)$
- Function $f : \mathcal{D}_1 \rightarrow \mathcal{D}_2$

$$\forall o_1, o_2 \in \mathcal{D}_1 : d_1(o_1, o_2) \approx d_2(f(o_1), f(o_2))$$

- Transformed distances need not be equal

Lower Bounding Metric Functions

- Bounds on transformations
- Exploitable by index structures

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

$$\forall o_1, o_2 \in \mathcal{D} : d_1(o_1, o_2) \leq d_2(o_1, o_2)$$

Lower Bounding Functions (cont.)

- Scaling factor
  - Some metric functions cannot be bounded.
  - We can bound them if they are reduced by a factor *s*

$$\exists 0 < s < 1 : \forall o_1, o_2 \in \mathcal{D} :$$

$$s \cdot d_1(o_1, o_2) \leq d_2(o_1, o_2)$$

  - $s \cdot d_1$ is a lower-bounding function of d_2
  - Maximum of all possible values of *s* is called the *optimal scaling factor.*

Example of Lower Bounding Functions

- L_p metrics
  - Any $L_{p'}$ metric is lower-bounding an L_p metric if p ≤ p'
  - Let $\vec{x}, \vec{y} \in \mathbb{R} \times \mathbb{R}$ are two vectors in a 2-D space

$$d_{L_2}(\vec{x}, \vec{y}) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

$$d_{L_1}(\vec{x}, \vec{y}) = |x_1 - y_1| + |x_2 - y_2|$$

  - L_1 is always bigger than L_2

Example of Lower Bounding Functions

- Quadratic Form Distance Function
  - Bounded by a scaled L_2 norm
  - Optimal scaling factor is

$$s_{optim} = \sqrt{\min_i \{\lambda_i\}}$$

where λ_i denote the eigenvalues of the quadratic form function matrix.

User-defined Metric Functions

- Different users have different preferences
  - Some people prefer car’s speed
  - Others prefer lower prices
  - etc…
- Preferences might be complex
  - Color histograms, data-mining systems
  - Can be learnt automatically
    - from the previous behavior of a user

User-defined Metric Functions

- Preferences expressed as another *distance function* d_u
  - 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
  - lower-bounds d_b and d_u
  - it is applied during the search
  - can exploit properties the index structure

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 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 d_1 distances

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

- Drawbacks
  - Must transform objects using the function *f*
  - False-positives
    - pruned using the original metric function

Embedding Examples

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

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

  - Transformation is very expensive
    - SparseMap extension reduces this cost

Embedding Examples

- Karhunen-Loeve tranformation
  - Linear transformation of vector spaces
  - Dimensionality reduction technique
    - Similar to Principal Component Analysis
  - Projects object *o* onto the first *k* < *n* basis vectors

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

  - Transformation is contractive
  - Used in the FastMap technique

Foundations of metric space searching

1. distance searching problem in metric spaces
2. metric distance measures
3. similarity queries
4. basic partitioning principles
5. principles of similarity query execution
6. policies to avoid distance computations
7. metric space transformations
8. **principles of approximate similarity search**
9. advanced issues

Principles of Approx. Similarity Search

- Approximate similarity search over-comes problems of exact similarity search when using traditional access methods.
  - 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 result sets satisfy the user's needs.

Principles of Approx. Similarity Search (cont.)

- Approximate similarity search processes a query faster at the price of imprecision in the returned result sets.
  - 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
    - Distances in the transformed space are smaller than in the original space.
    - Possible false hits
  - Example:
    - Dimensionality reduction techniques such as
    - KLT, DFT, DCT, DWT
    - VA-files
  - We will not discuss this approximation strategy in details.

Basic Strategies (cont.)

- Reducing subsets of data to be examined
  - Not promising data is not accessed.
  - False dismissals can occur.
  - This strategy will be discussed more 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

- Exact similarity search algorithms are
  - Iterative processes, where
  - Current result set is improved at each step.

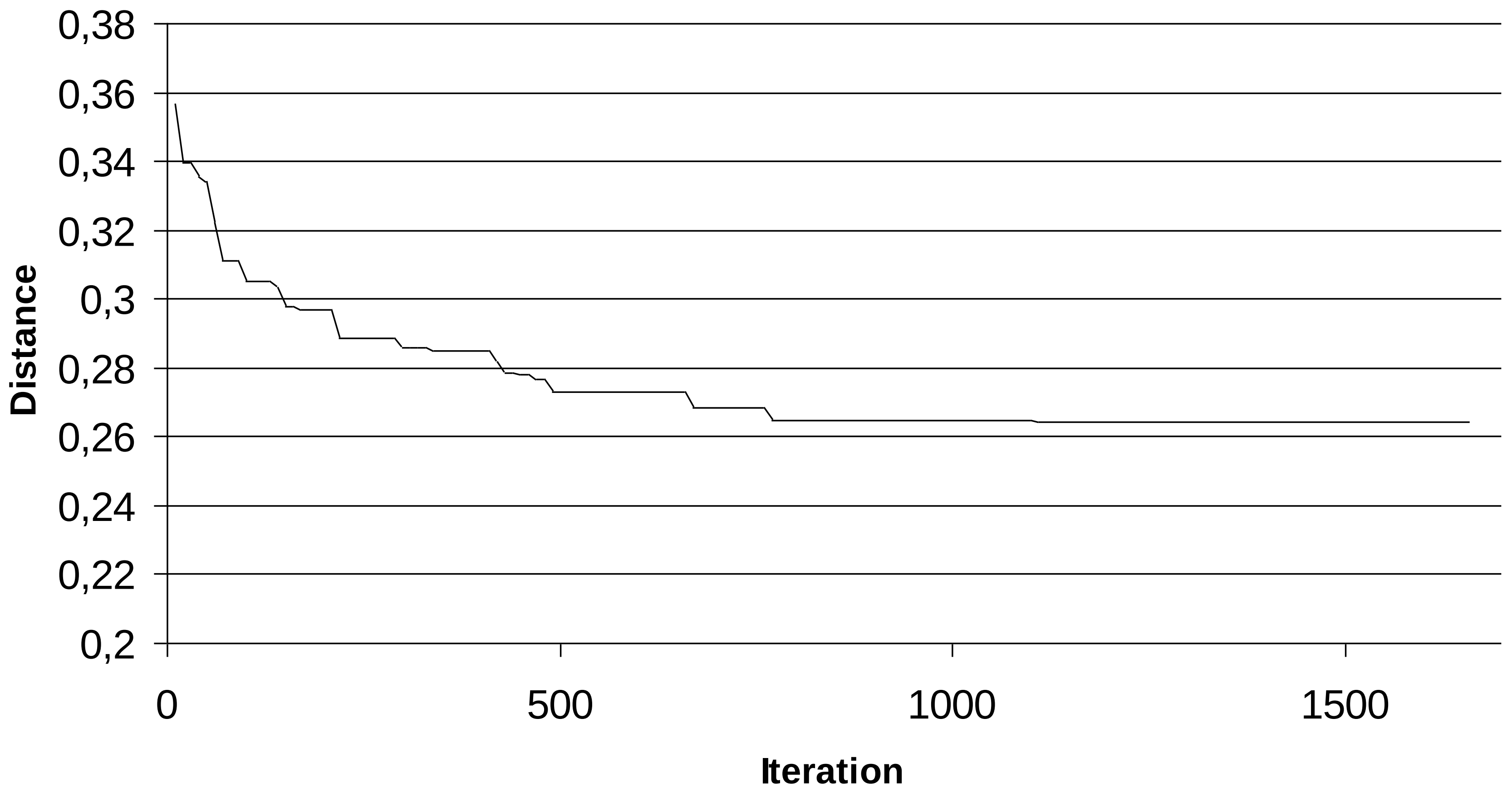
- Exact similarity search algorithms stop
  - When no further improvement is possible.

Early Termination Strategies (cont.)

- Approximate similarity search algorithms
  - Use a “relaxed” *stop condition* that
  - 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.

Early Termination Strategies (cont.)

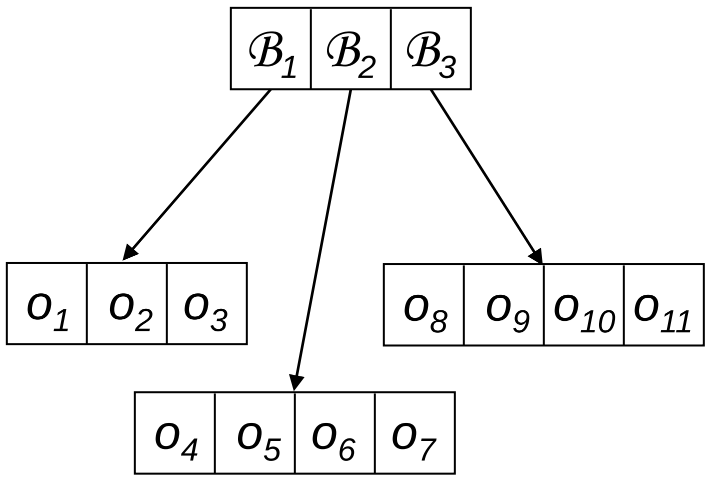


Relaxed Branching Strategies

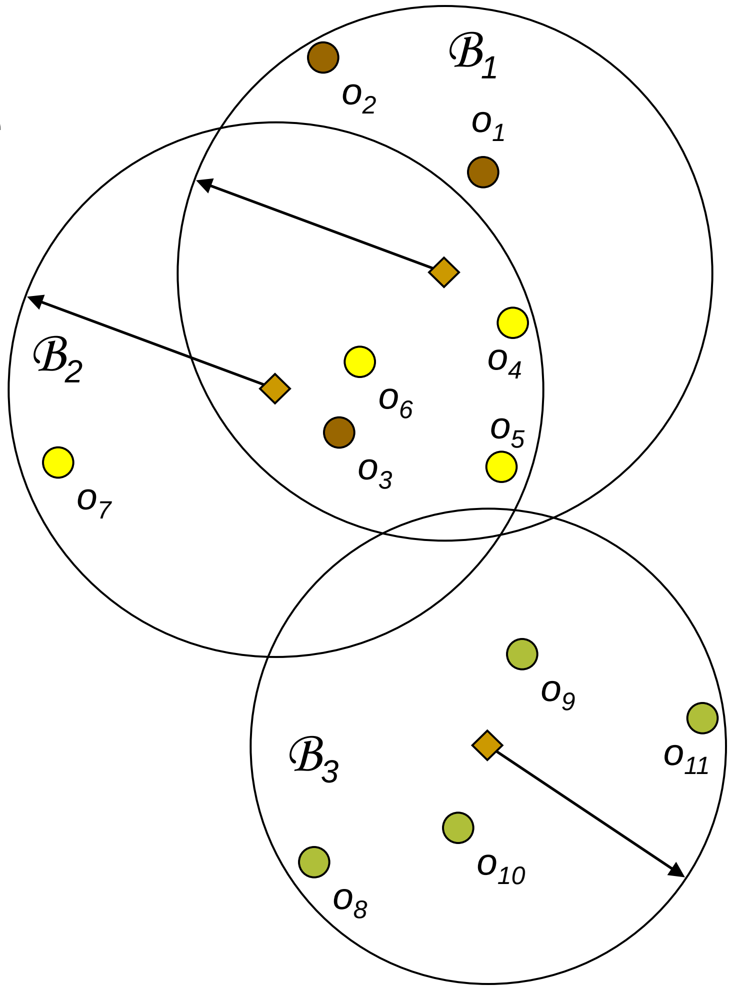
- Exact similarity search algorithms
  - Access all data regions overlapping the query region and discard all the others.
- Approximate similarity search algorithms
  - Use a “relaxed” *pruning condition* that
  - Rejects regions overlapping the query region when it detects a low likelihood that data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

Approximate Search: Example

- A hypothetical index structure
  - Three ball regions

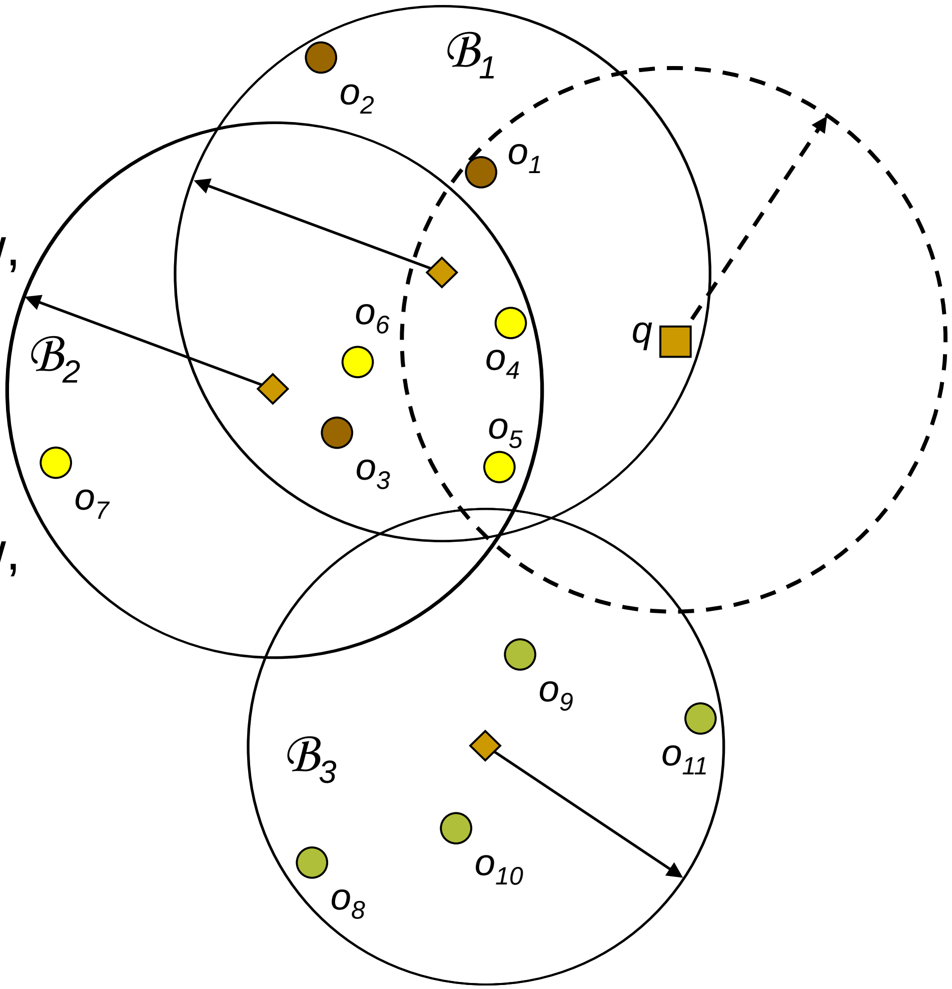


$\mathcal{B}_1$ $\mathcal{B}_2$ $\mathcal{B}_3$
o_1 o_2 o_3
o_8 o_9 o_{10} o_{11}
o_4 o_5 o_6 o_7



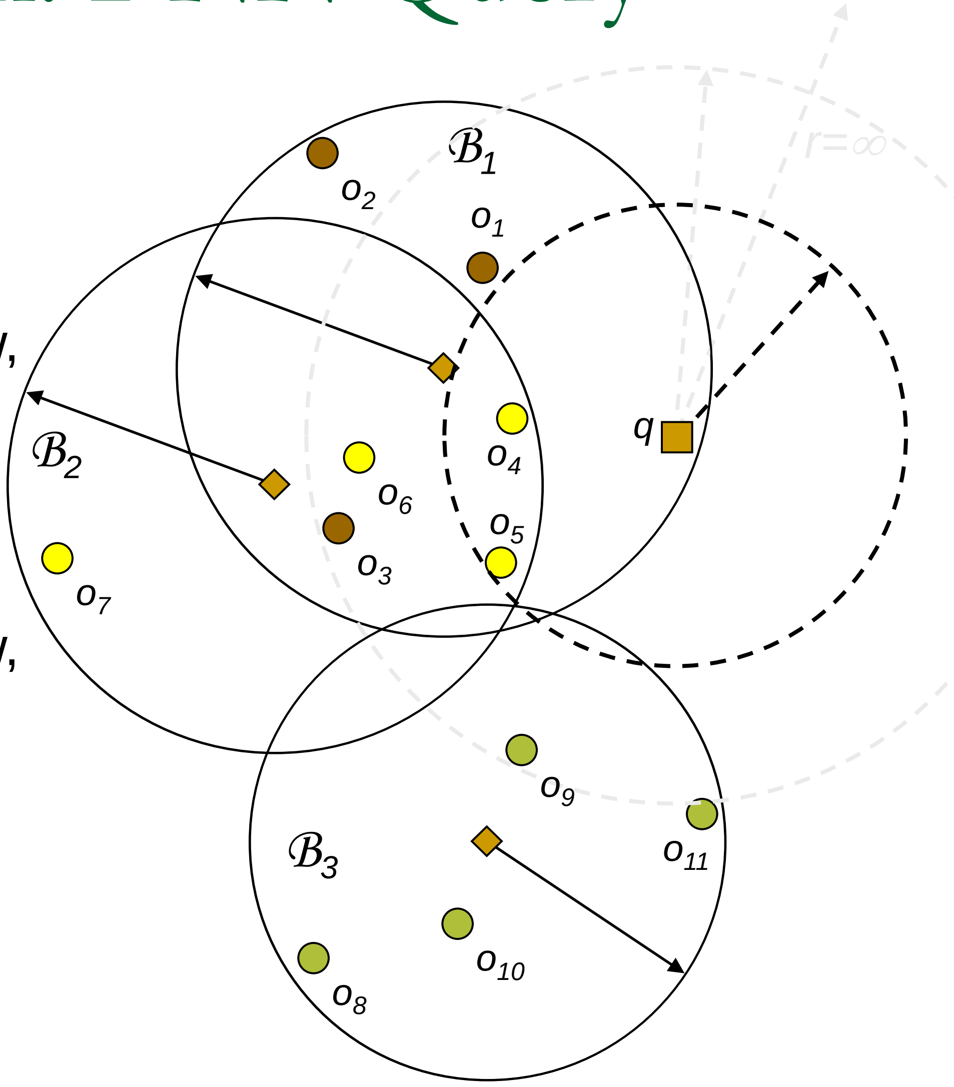
Approximate Search: Range Query

- Given a range query:
- Access $\mathcal{B}_1$
  - Report o_1
  - If early termination stopped now, we would loose objects.
- Access $\mathcal{B}_2$
  - Report o_4 ,o_5
  - If early termination stopped now, we would not loose anything.
- Access $\mathcal{B}_3$
  - Nothing to report
  - A relaxed branching strategy may discard this region – we don't loose 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 loose objects.
- Access $\mathcal{B}_2$
  - Neighbors: o_4, o_5
  - If early termination stopped now, we would not loose anything.
- Access $\mathcal{B}_3$
  - Neighbors: o_4, o_5 – no change
  - A relaxed branching strategy may discard this region – we don't loose anything.



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 cost of accuracy in the results.
- Good approximate search algorithms should
  - offer high improvement in efficiency with high accuracy in the results.

Measures of Performance: 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.

Improvement in Efficiency (cont.)

- *IE*=10 means that approximate execution is 10 times faster
- Example:
  - exact execution 6 minutes
  - approximate execution 36 seconds

Measures of Performance: Precision and Recall

- Widely used in Information Retrieval as a performance assessment.
- *Precision*: ratio between the retrieved qualifying objects and the total objects retrieved.
- *Recall*: ratio between the retrieved qualifying objects and the total qualifying objects.

Precision and Recall (cont.)

- Accuracy can be quantified with *Precision (P)* and *Recall (R)*:

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

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

Precision and Recall (cont.)

- They are very intuitive but in our context
  - Their interpretation is not obvious & misleading!!!
- For approximate range search we typically have $S^A \subseteq S$
  - Therefore, precision is always 1 in this case
- Results of *k-NN(q)* have always size *k*
  - Therefore, precision is always equal to recall in this case.
- Every element has the same importance
  - Loosing the first object rather than the 1000th one is the same.

Precision and Recall (cont.)

- Suppose a *10-NN*(*q*):
  - $S=\{1,2,3,4,5,6,7,8,9,10\}$
  - $S^{A1}=\{2,3,4,5,6,7,8,9,10,11\}$ the object *1* is missing
  - $S^{A2}=\{1,2,3,4,5,6,7,8,9,11\}$ the object *10* is missing
- In both cases: $P = R = 0.9$
  - However S^{A2} can be considered better than S^{A1}.

Precision and Recall (cont.)

- Suppose *1-NN*(*q*):
  - $S=\{1\}$
  - $S^{A1}=\{2\}$ just one object was skipped
  - $S^{A2}=\{10000\}$ the first 9,999 objects were skipped
- In both cases: $P = R = 0$
  - However S^{A1} can be considered much better than S^{A2}.

Measures of Performance: Relative Error on Distances

- Another possibility to assess the accuracy is the use of the *relative error on distances (ED)*
  - It compares the distances from a query object to objects in the approximate and exact results

$$ED = \frac{d(o^A,q) - d(o^N,q)}{d(o^N,q)} = \frac{d(o^A,q)}{d(o^N,q)} - 1$$

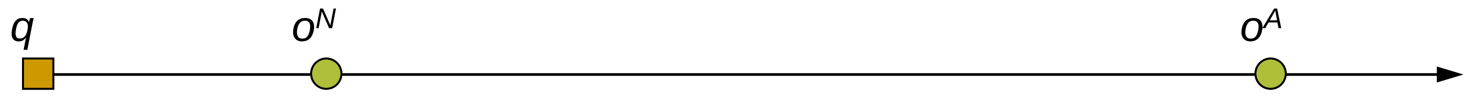
where o^A and o^N are the approximate and the actual nearest neighbors, respectively.

- Generalisation to the case of the *j*-th *NN*:

$$ED_j = \frac{d(o_j^A,q)}{d(o_j^N,q)} - 1$$

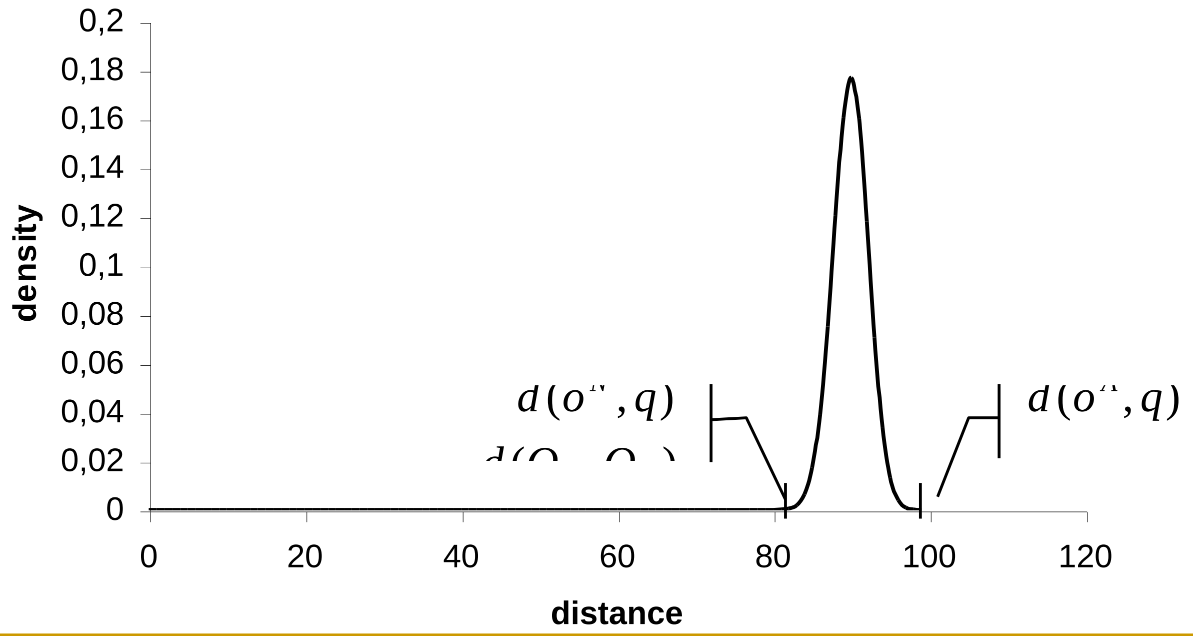
Relative Error on Distances (cont.)

- It has a drawback:
  - It does not take the distribution of distances into account.
- Example1: The difference in distance from the query object to o^N and o^A is large (compared to the range of distances)
  - If the algorithm misses o^N and takes o^A, *ED* is large even if just one object has been missed.



Relative Error on Distances (cont.)

- Example 2: Almost all objects have the same (large) distance from the query object.
  - Choosing the farthest rather than the nearest neighbor would produce a small *ED,* even if almost all objects have been missed.



Measures of Performance: Error on Position

- Accuracy can also be measured as the *Error on Position (EP)*
  - i.e., the discrepancy between the ranks in approximate and exact results.
- Obtained using the *Sperman Footrule Distance (SFD)*:

$$SFD = \sum_{i=1}^{|X|} \left| S_1(o_i) - S_2(o_i) \right|$$

$|X|$ – the dataset's cardinality

$S_i(o)$ – the position of object o in the ordered list S_i

Error on Position (cont.)

- SFD computes correlation of two ordered lists.
  - Requires both the lists to have identical elements.
- For partial lists: *Induced Footrule Distance (IFD):*

$$IFD = \sum_{i=1}^{|S^A|} \left| OX(o_i) - S^A(o_i) \right|$$

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

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

Error on Position (cont.)

- Position in the approximate result is always smaller than or equal to the one in the exact result.
  - S^A is a sub-lists of OX
  - $S^A(o) \leq OX(o)$
- A normalisation factor $|S^A| \cdot |X|$ can also be used
- The *error on position (EP)* is defined as

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

Error on Position (cont.)

- Suppose |X|=10,000
- Let us consider a *10-NN*(*q*):
  - *S*={*1,2,3,4,5,6,7,8,9,10*}
  - S^{A1}={*2,3,4,5,6,7,8,9,10,11*} the object *1* is missing
  - S^{A2}={*1,2,3,4,5,6,7,8,9,11*} the object *10* is missing
- As also intuition suggests:
  - In case of S^{A1}, $EP = 10 / (10 \cdot 10{,}000) = 0.0001$
  - In case of S^{A2}, $EP = 1 / (10 \cdot 10{,}000) = 0.00001$

Error on Position (cont.)

- Suppose |X|=10,000
- Let us consider a *1-NN*(*q*):
  - $S=\{1\}$
  - $S^{A1}=\{2\}$ just one object was skipped
  - $S^{A2}=\{10{,}000\}$ the first 9,999 objects were skipped
- As also intuition suggests :
  - In case of S^{A1}, $EP = (2-1)/(1 \cdot 10{,}000) = 1/(10{,}000) = 0.0001$
  - In case of S^{A2}, $EP = (10{,}000-1)/(1 \cdot 10{,}000) = 0.9999$

Foundations of metric space searching

1. distance searching problem in metric spaces
2. metric distance measures
3. similarity queries
4. basic partitioning principles
5. principles of similarity query execution
6. policies to avoid distance computations
7. metric space transformations
8. principles of approximate similarity search
9. **advanced issues**

Statistics on Metric Datasets

- Statistical characteristics of datasets form the basis of performance optimisation in databases.
- Statistical information is used for
  - Cost models
  - Access structure tuning
- Typical statistical information
  - Histograms of frequency values for records in databases
  - Distribution of data, in case of data represented in a vector space

Statistics on Metric Datasets (cont.)

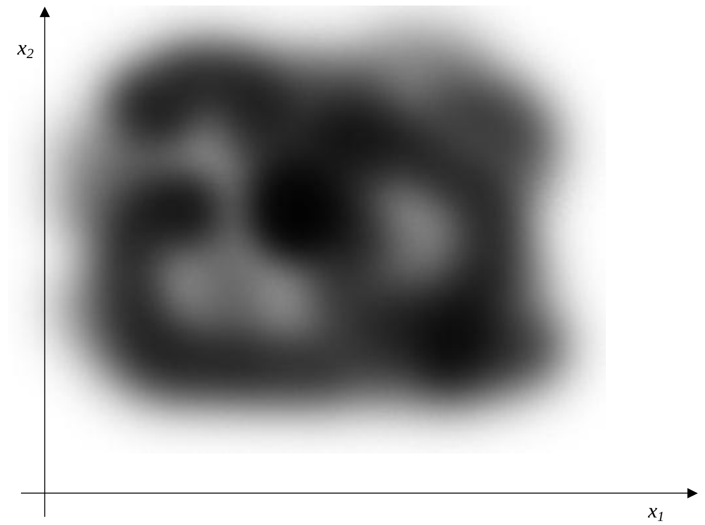
- Histograms and data distribution cannot be used in generic metric spaces
  - We can only rely on distances
  - No coordinate system can be used
- Statistics useful for techniques for similarity searching in metric spaces are
  - Distance density and distance distribution
  - Homogeneity of viewpoints
  - Proximity of ball regions

Data Density vs. Distance Density

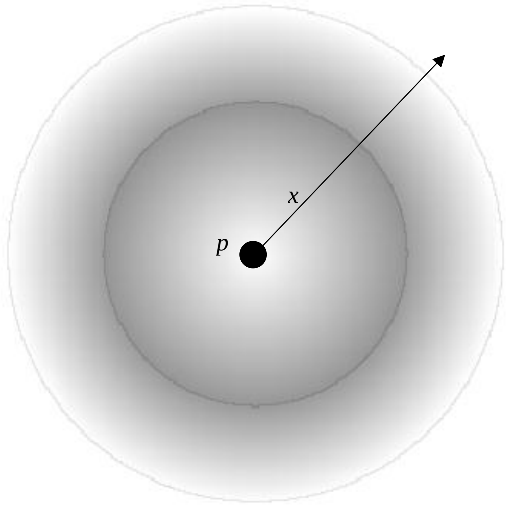
- *Data density* (applicable just in vector spaces)
  - characterizes how data are placed in the space
  - coordinates of objects are needed to get their position
- *Distance density* (applicable in generic metric spaces)
  - characterizes distances among objects
  - no need of coordinates
  - just a distance functions is required

Data Density vs. Distance Density (cont.)

- Data density



- Distance density from the object *p*



Distance Distribution and Distance Density

- The *distance distribution* with respect to the object *p* (*viewpoint*) is

$$F_{D_p}(x) = \Pr\{D_p \le x\} = \Pr\{d(p,o) \le x\}$$

  where D_p is a random variable corresponding to the distance $d(p,o)$ and *o* is a random object of the metric space.

- The *distance density* from the object *p* can be obtained as the derivative of the distribution.

Distance Distribution and Distance Density (cont.)

- The *overall distance distribution* (informally) is the probability of distances among objects

$$F(x) = \Pr\{d(o_1, o_2) \le x\}$$

where o_1 and o_2 are random objects of the metric space.

Homogeneity of Viewpoints

- A viewpoint (distance distribution from *p*) is different from another viewpoint.
  - Distances from different objects are distributed differently.
- A viewpoint is different from the overall distance distribution.
  - The overall distance distribution characterize the entire set of possible distances.
- However, the overall distance distribution can be used in place of any viewpoint if the dataset is probabilistically homogeneous.
  - i.e., when the discrepancy between various viewpoints is small.

Homogeneity of Viewpoints (cont.)

- The index of *Homogeneity of Viewpoints* (*HV*) for a metric space $\mathcal{M}=(\mathcal{D},d)$ is:

$$HV(\mathcal{M}) = 1 - \operatorname*{avg}_{p_1,p_2 \in \mathcal{D}} \delta(F_{p_1}, F_{p_2})$$

where p_1 and p_2 are random objects and the discrepancy between two viewpoints is

$$\delta(F_{p_i}, F_{p_j}) = \operatorname*{avg}_{x \in [0,d^+]} \left| F_{p_i}(x) - F_{p_j}(x) \right|$$

where F_{pi} is the viewpoint of p_i

Homogeneity of Viewpoints (cont.)

- If $HV(\mathcal{M}) \approx 1$, the overall distance distribution can be reliably used to replace any viewpoint.

Proximity of Ball Regions

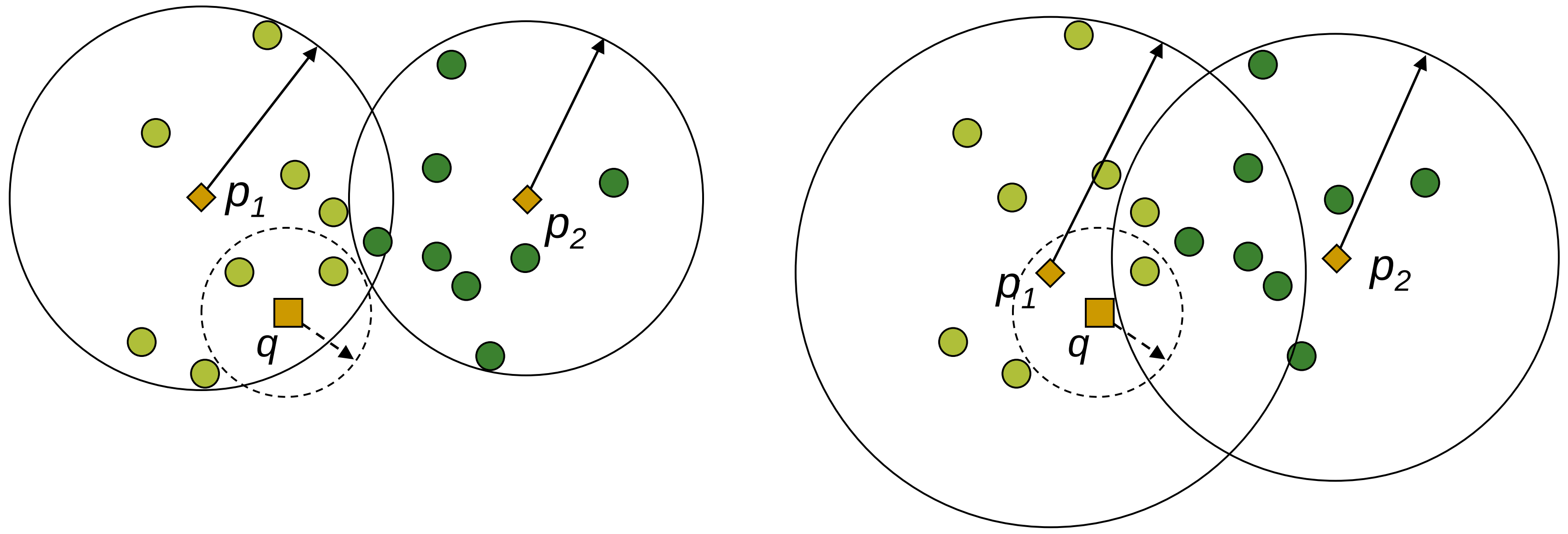
- Proximity of two regions is a measure that estimates the number of objects contained in their overlap
- Used in:
  - Region splitting for partitioning
    - After splitting one region, the new regions should share as little objects as possible.
  - Disk allocation
    - Enhancing performance by distributing data over several disks.
  - Approximate search
    - Applied in relaxed branching strategy – a region is accessed if there is high probability to have objects in the intersection.

Proximity of Ball Regions (cont.)

- In Euclidean spaces, it is easy to obtain
  - compute data distributions
  - compute integrals of data distribution on regions' intersection
- In metric spaces
  - coordinates cannot be used
    - data distribution cannot be exploited
  - *distance density/distribution* is the only available statistical information

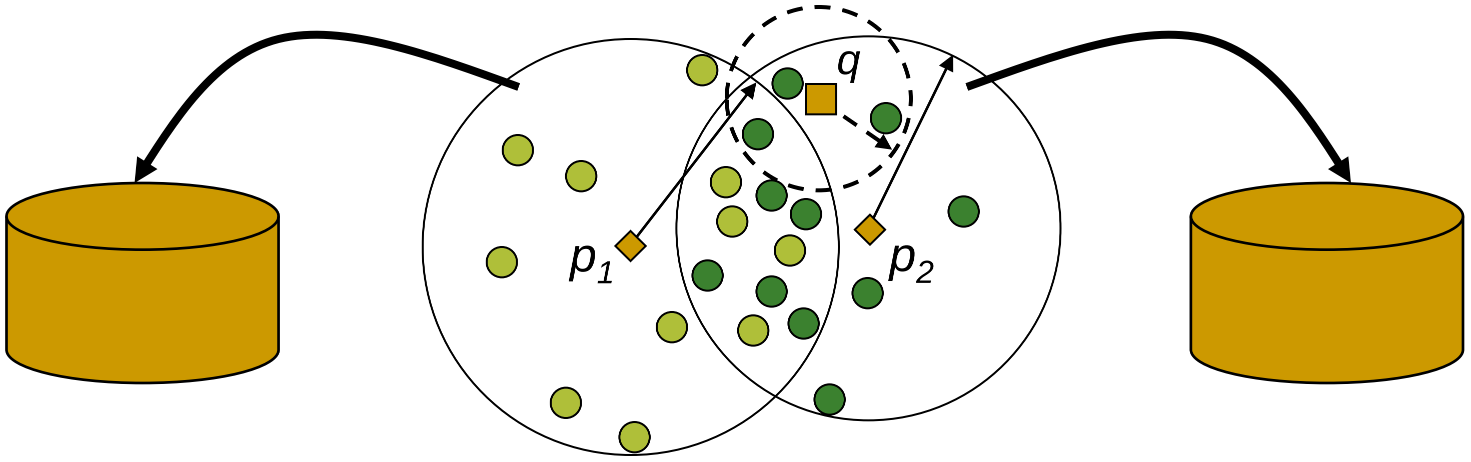
Proximity of Ball Regions: Partitioning

- Queries usually follow data distribution
- Partition data to avoid overlaps, i.e. accessing both regions.
    - Low overlap (left) vs. high overlap (right)



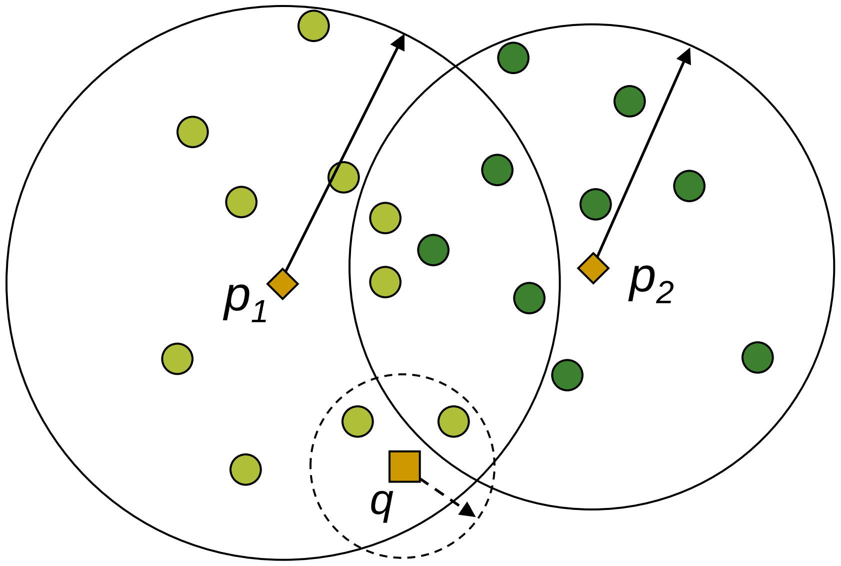
Proximity of Ball Regions: Data Allocation

- Regions sharing many objects should be placed on different disk units – declustering
  - Because there is high probability of being accessed together by the same query.



Proximity of Ball Regions: Approximate Search

- Skip visiting regions where there is low chance to find objects relevant to a query.



Proximity of Metric Ball Regions

- Given two ball regions $\mathcal{R}_1=(p_1,r_1)$ and $\mathcal{R}_2=(p_2,r_2)$, we define *proximity* as follows:

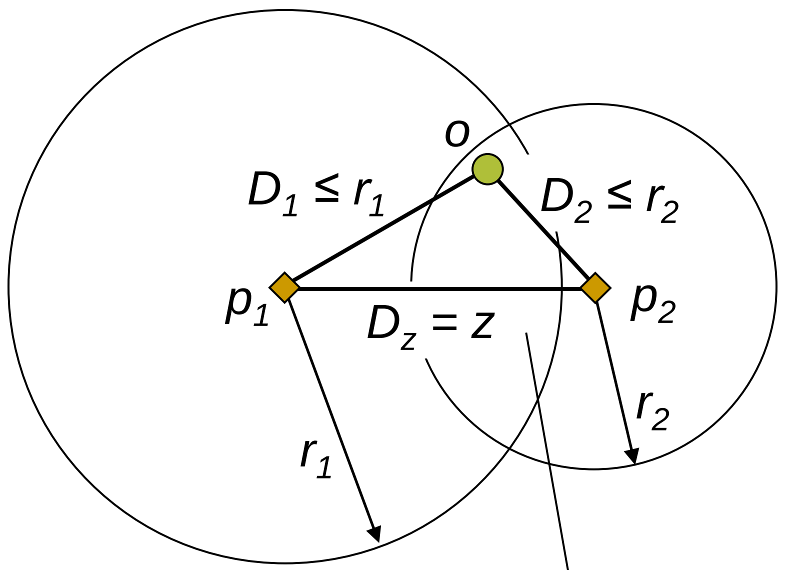
$$prox(\mathcal{R}_1,\mathcal{R}_2)=\Pr\{d(p_1,o)\leq r_1 \wedge d(p_2,o)\leq r_2\}$$

- In real-life datasets, distance distribution does not depend on specific objects
  - Real datasets have a high index of homogeneity.
- We define the *overall proximity*

$$prox_z(r_1,r_2)=\Pr\{d(p_1,o)\leq r_1 \wedge d(p_2,o)\leq r_2 \mid d(p_1,p_2)=z\}$$

Proximity of Metric Ball Regions (cont.)

- Overall proximity:



Triangle inequality:

$D_z \le D_1 + D_2$

Proximity: Probability that an object *o* appears in the intersection.

Proximity: Computational Difficulties

- Let $D_1 = d(p_1, o)$, $D_2 = d(p_2, o)$, $D_z = d(p_1, p_2)$ be random variables, the *overall proximity* can be mathematically evaluated as

$$prox_z(r_1, r_2) = \int_0^{r_1} \int_0^{r_2} f_{D_1, D_2 | D_z}(x, y \mid d_z) dydx$$

- An analytic formula for the joint conditional density $f_{D_1, D_2 | D_z}$ is not known for generic metric spaces.

Proximity: Computational Difficulties (cont.)

- Idea: Replace the *joint conditional density* $f_{D1,D2|Dz}(x,y|z)$ with the *joint density* $f_{D1,D2}(x,y)$.
  - However, these densities are different.
  - The joint density is easier to obtain:

$$f_{D_1 D_2}(x,y) = f_{D_1}(x) \cdot f_{D_2}(y)$$

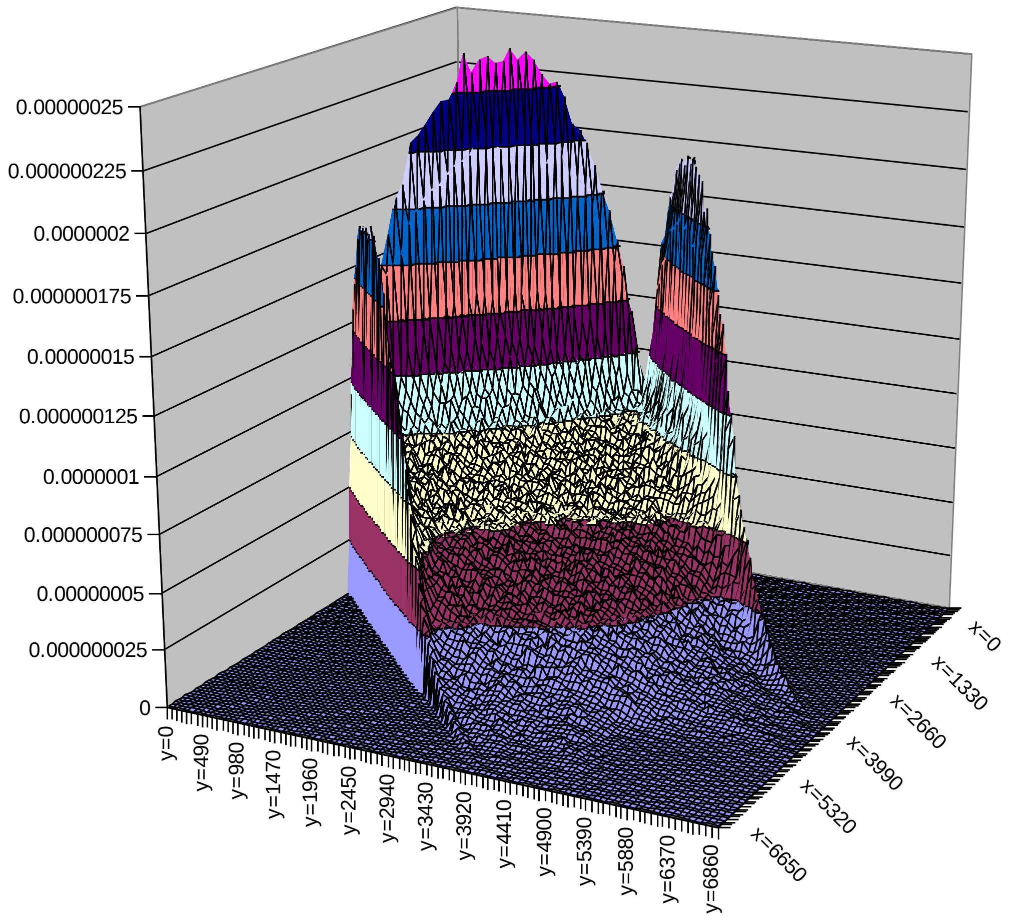
    - If the overall density is used:

$$f_{D_1 D_2}(x,y) = f(x) \cdot f(y)$$

- The original expression can only be approximated.

Proximity: Considerations (cont.)

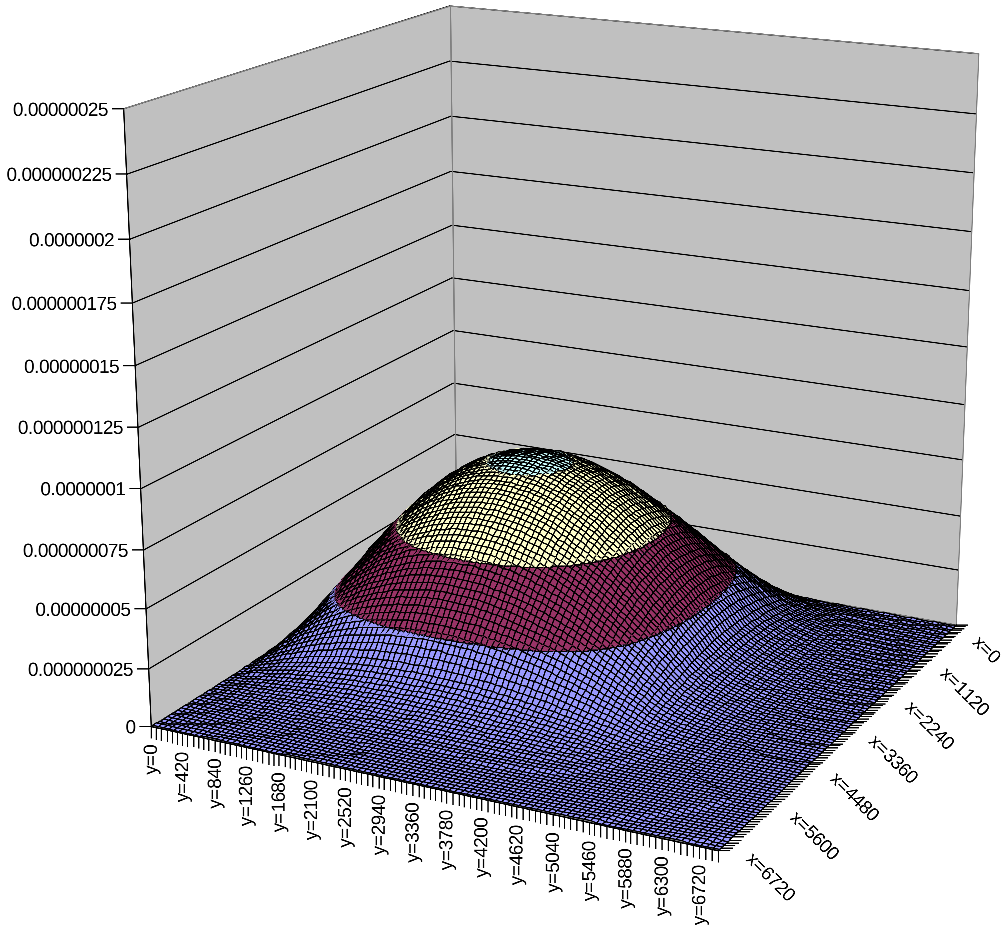
- The joint conditional density is zero
  - When *x*,*y* and *z* do not satisfy the triangle inequality.
  - Simply such distance cannot exist in metric space.



joint conditional density

Proximity: Considerations (cont.)

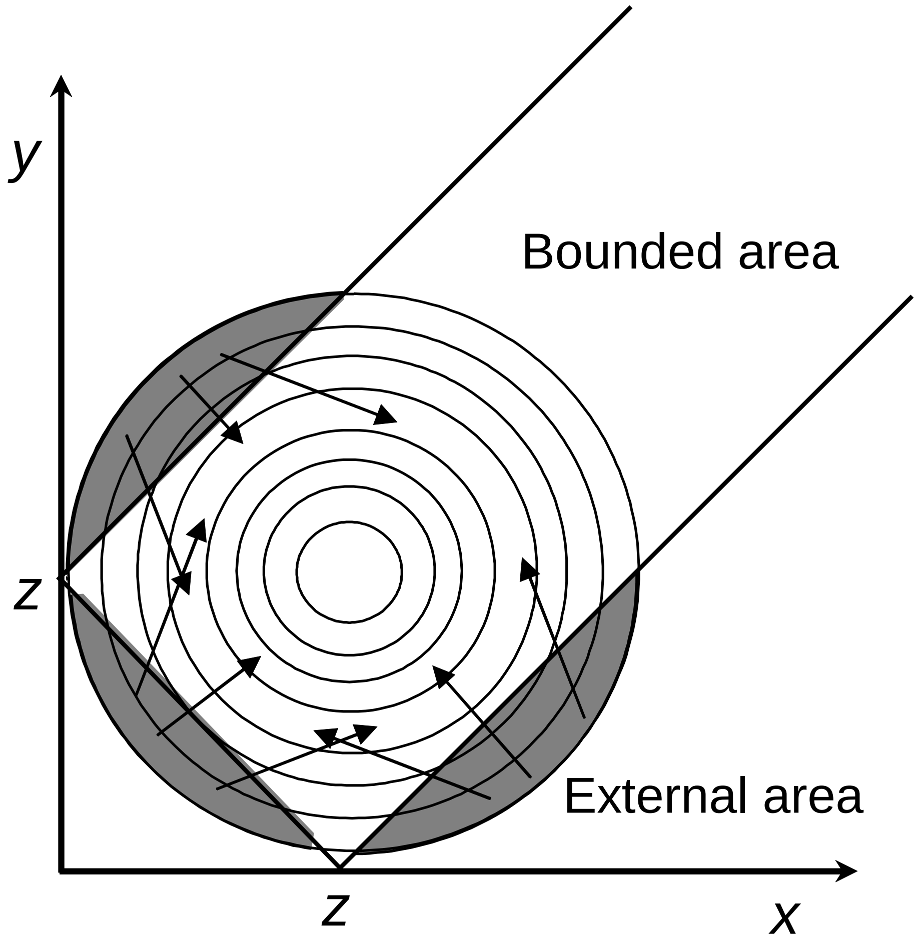
- The joint density is not restricted
  - Idea: the joint conditional density is obtained by dragging values out of borders of the triangle inequality to the border.



joint density

Proximity: Approximation

- Proximity can be computed in *O*(*n*) with high precision
  - *n* is the number of samples for the integral computation of *f*(*x*).
  - *Distance density and distribution* are the only information that need to be pre-computed and stored.



Performance Prediction

- Distance distribution can be used for performance prediction of similarity search access methods
  - Estimate the number of accessed subsets
  - Estimate the number of distance computations
  - Estimate the number of objects retrieved

- Suppose a dataset was partitioned in *m* subsets
- Suppose every dataset is bounded by a ball region $\mathcal{R}_i=(p_i,r_i)$, $1\leq i \leq m$, with the pivot p_i and radius r_i

Performance Prediction: Range Search

- A range query $R(q,r_q)$ will access a subset bounded by the region $\mathcal{R}_i$ if it intersects the query
  - i.e., if $d(q,p_i) \leq r_i+r_q$
- The probability for a *random region* $\mathcal{R}^r=(p,r)$ to be accessed is

$$\Pr\{d(q,p) \leq r + r_q\} = F_q(r + r_q) \approx F(r + r_q)$$

where p is the random centre of the region, F_q is the q's viewpoint, and the dataset is highly homogeneous.

Performance Prediction: Range Search (cont.)

- The expected number of accessed subsets is obtained by summing the probability of accessing each subset:

$$subsets(R(q, r_q)) \approx \sum_{i=1}^{m} F(r_i + r_q)$$

provided that we have a data structure to maintain the r_i's.

Performance Prediction: Range Search (cont.)

- The expected number of distance computations is obtained by summing the size of subsets and using the probability of accessing as a weight

$$distances(R(q,r_q)) \approx \sum_{i=1}^{m} |\mathcal{R}_i| F(r_i + r_q)$$

- The expected size of the result is given simply as

$$objects(R(q,r_q)) \approx n \cdot F(r_q)$$

where *n* is the cardinality of the entire dataset.

Performance Prediction: Range Search (cont.)

- Data structure to maintain the *radii* and the *cardinalities* of all bounding regions in needed
  - The size of this information can become unacceptable – grows linearly with the size of the dataset.

Performance Prediction: Range Search (cont.)

- Previous formulas can be reliably approximated by using the average information on each level of a tree (more compact)

$$subsets(R(q,r_q)) \approx \sum_{l=1}^{L} M_l F(ar_l + r_q)$$

$$distances(R(q,r_q)) \approx \sum_{l=1}^{L} M_{l+1} F(ar_l + r_q)$$

where M_l is the number of subsets at level l, ar_l is the average covering radius at level l, and L is the total number of levels.

Performance Prediction: *k*-NN Search

- The optimal algorithm for *k-NN*(*q*) would access all regions that intersect $\mathcal{R}(q,d(q,o_k))$, where o_k is the *k*-th nearest neighbor of *q.*
  - The cost would be equal to that of the range query $R(q,d(q,o_k))$
  - However $d(q,o_k)$ is not known in advance.
  - The distance density of o_k (f_{Ok}) can be used instead

$$F_{o_k}(x) = \Pr\{d(q,o_k) \le x\} = 1 - \sum_{i=0}^{k-1} \binom{n}{i} F(x)^i (1 - F(x))^{n-i}$$

  - The density f_{Ok} is the derivative of F_{Ok}

Performance Prediction: *k-NN* Search (cont.)

- The expected number of accessed subsets is obtained by integrating the cost of a range search multiplied by the density of the *k*-th *NN* distance

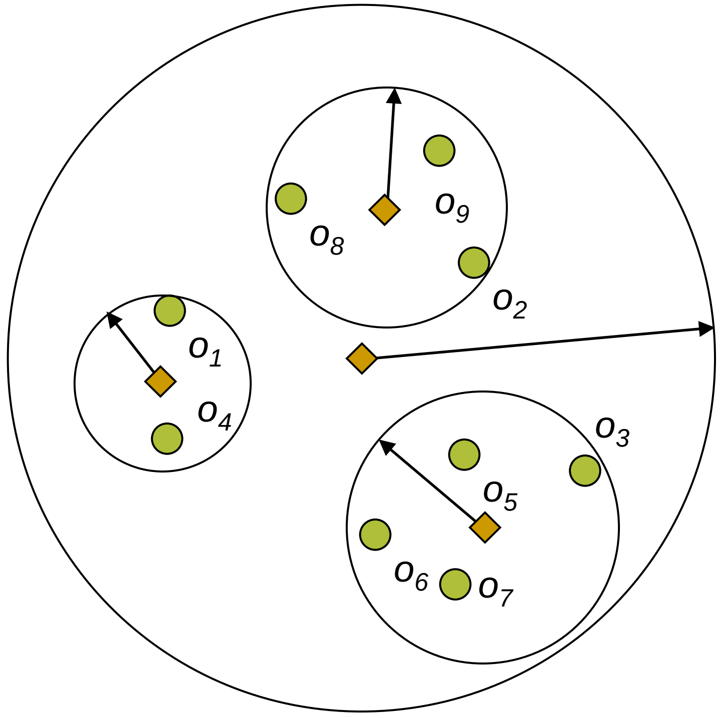
$$subsets(kNN(q)) \approx \int_0^{d^+} subsets(R(q,r))\, f_{o_k}(r)\, dr$$

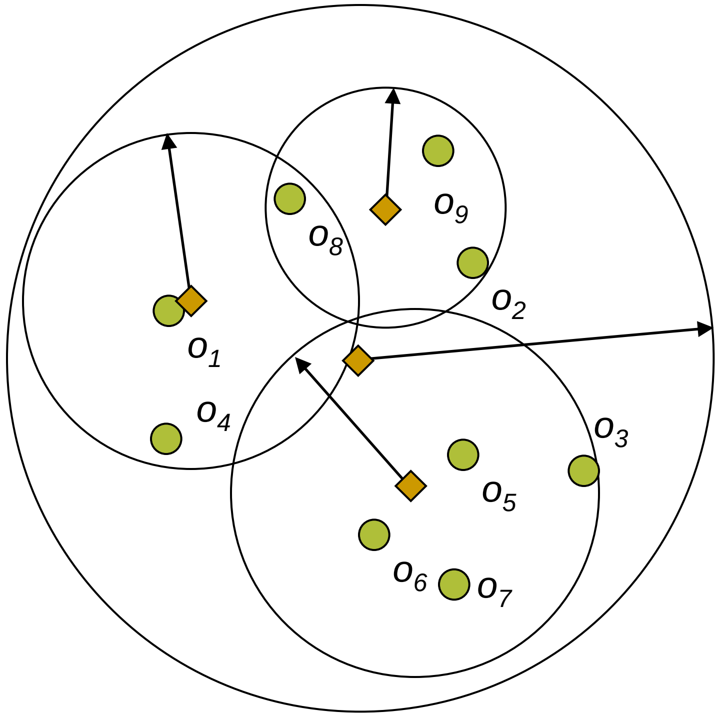
- Similarly, the expected number of distance computations is

$$distances(kNN(q)) \approx \int_0^{d^+} distances(R(q,r))\, f_{o_k}(r)\, dr$$

Tree Quality Measures

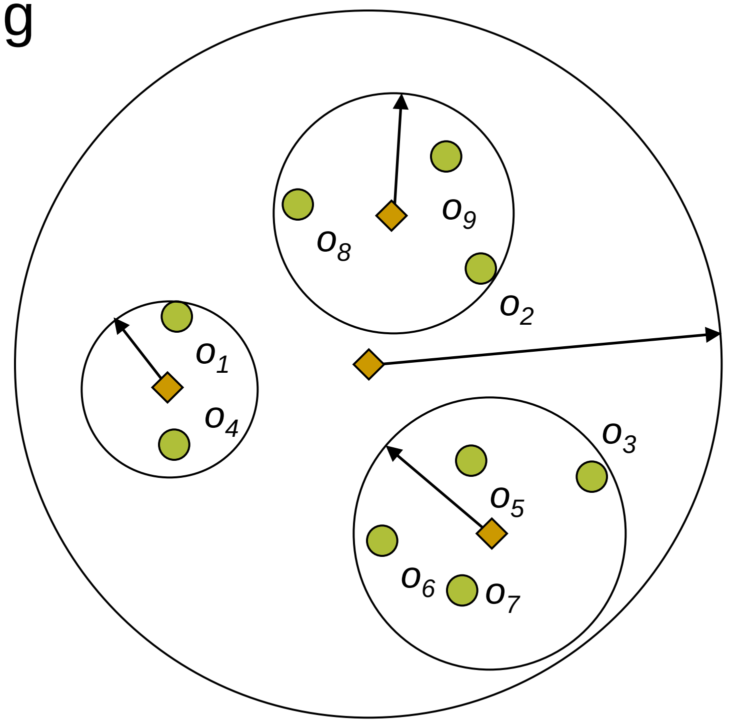
- Consider our hypothetical index structure again
- We can build two different trees over the same dataset





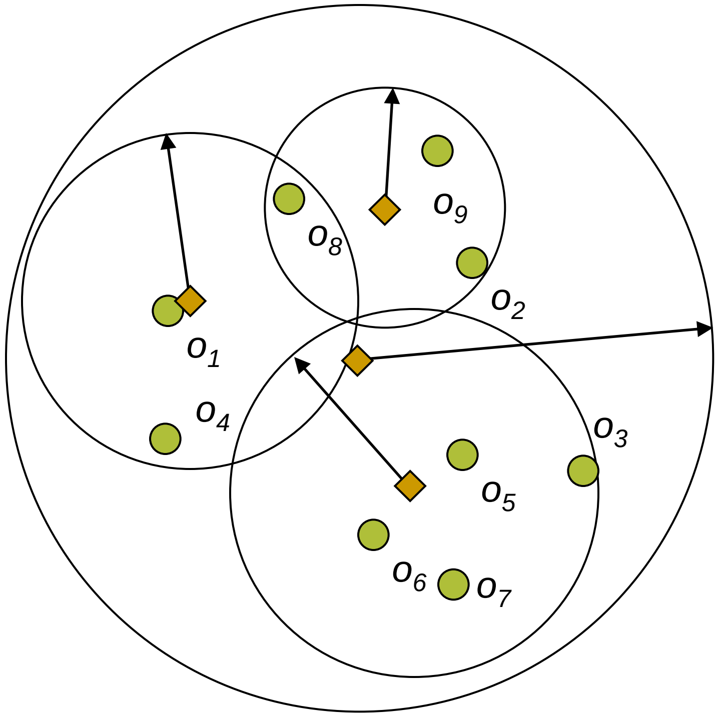
Tree Quality Measures (cont.)

- The first tree is more compact.
  - Occupation of leaf nodes is higher.
  - No intersection between covering regions.



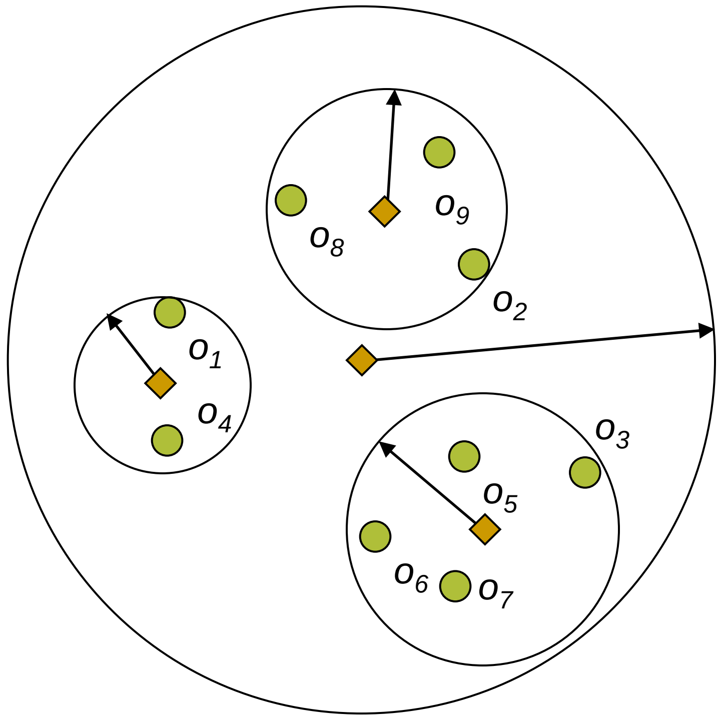
Tree Quality Measures (cont.)

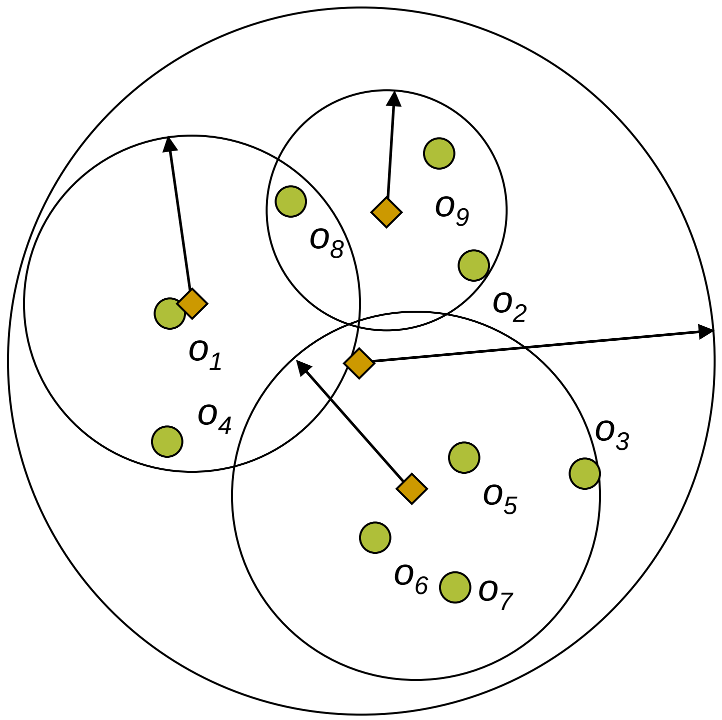
- The second tree is less compact.
  - It may result from deletion of several objects.
  - Occupation of leaf nodes is poor.
  - Covering regions intersect.
  - Some objects are in the intersection.



Tree Quality Measures (cont.)

- The first tree is ‘better’!
- We would like to measure quality of trees.





Tree Quality Measures: Fat Factor

- This quality measure is based on overlap of metric regions.

$$overlap = \frac{|R_1 \cap R_2|}{|R_1 \cup R_2|}$$

- Different from the previous concept of overlap estimation.
  - It is more local.
  - Number of objects in the overlap divided by the total number of objects in both the regions.

Fat Factor (cont.)

- "Goodness" of a tree is strictly related to overlap.
  - Good trees are with overlaps as small as possible.
- The measure counts the total number of node accesses required to answer exact match queries for all database objects.
  - If the overlap of regions $\mathcal{R}_1$ and $\mathcal{R}_2$ contains o, both corresponding nodes are accessed for $R(o,0)$.

Absolute Fat Factor: definition

- Let T be a metric tree of n objects with height h and $m \geq 1$ nodes. The *absolute fat-factor* of T is:

$$fat(T) = \frac{I_C - nh}{n} \cdot \frac{1}{m-h}$$

- I_C – total number of nodes accessed during n exact match query evaluations: from nh to nm

Absolute Fat Factor: Example

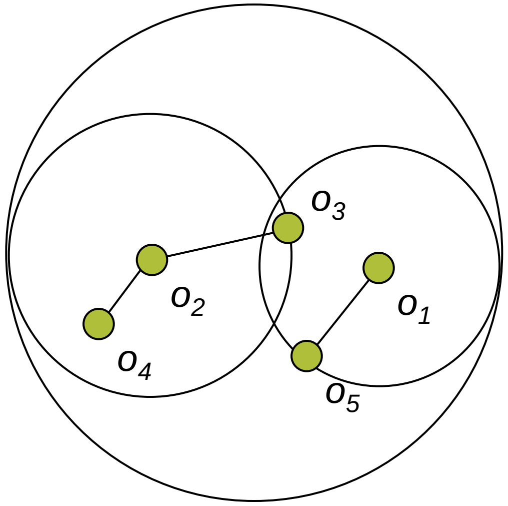
- An ideal tree needs to access just one node per level.
  - $fat(T_{ideal}) = 0 \quad \rightarrow \quad I_C = nh$
- The worst tree always access all nodes.
  - $fat(T_{worst}) = 1 \quad \rightarrow \quad I_C = nm$

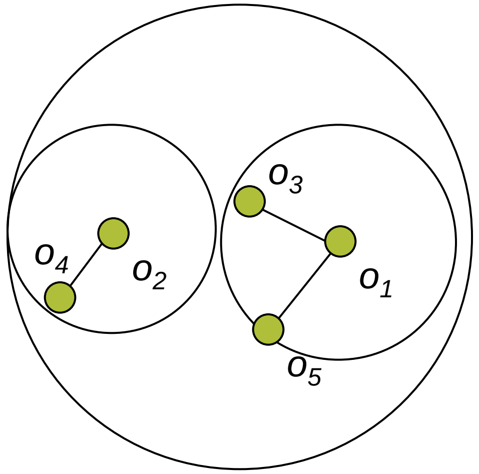
Absolute Fat Factor: Example

- Two trees organizing 5 objects:
  - $n=5 \quad m=3 \quad h=2$

- $I_C=11$
- $fat(T)=0.2$

- $I_C=10$
- $fat(T)=0$





Absolute Fat Factor: Summary

- Absolute fat-factor’s consequences:
  - Only range queries taken into account
    - *k-NN* queries are special case of range queries
  - Distribution of exact match queries follows distribution of data objects
    - In general, it is expected that queries are issued in dense regions more likely.
- The number of nodes in a tree is not considered.
  - A big tree with a low fat-factor is better than a small tree with the fat-factor a bit higher.

Relative Fat Factor: Definition

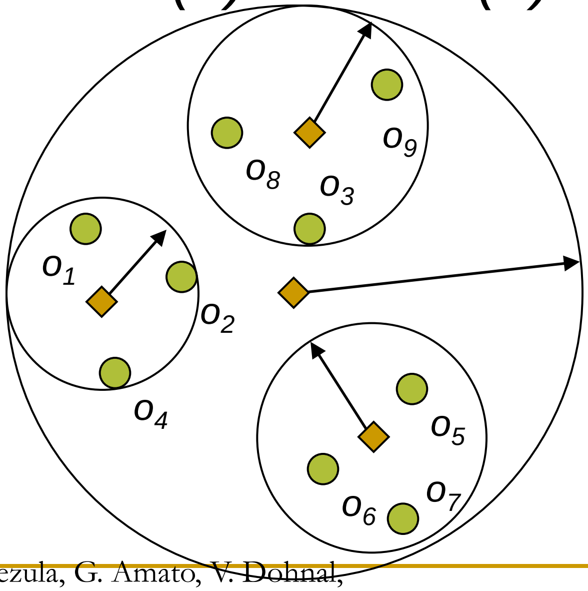
- Penalizes trees with more than minimum number of nodes.
- Let *T* be a metric tree with more than one node organizing *n* objects. The *relative fat-factor* of *T* is defined as:

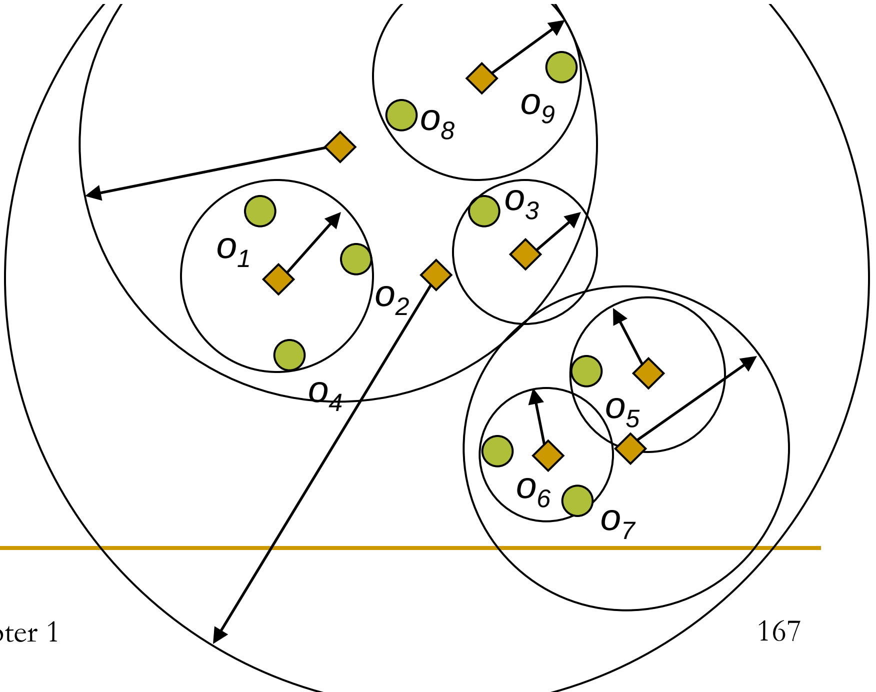
$$rfat(T) = \frac{I_C - nh_{\min}}{n} \cdot \frac{1}{m_{\min} - h_{\min}}$$

  - I_C – total number of nodes accessed
  - *C* – capacity of a node in objects
  - Minimum height: $h_{\min} = \lceil \log_C n \rceil$
  - Minimum number of nodes: $m_{\min} = \sum_{i=1}^{h_{\min}} \left\lceil n / C^i \right\rceil$

Relative Fat Factor: Example

- Two trees organizing 9 objects:
  - $n=9$ $C=3$ $h_{min}=2$ $m_{min}=4$
- Minimum tree
  - $I_C=18$ $h=2$ $m=4$
  - $rfat(T)=0$ $fat(T)=0$
- Non-optimal tree
  - $I_C=27$ $h=3$ $m=8$
  - $rfat(T)=0.5$ $fat(T)=0$





Tree Quality Measures: Conclusion

- Absolute fat-factor
  - *$0 \leq fat(T) \leq 1$*
  - Region overlaps on the same level are measured.
  - Under-filled nodes are not considered.
  - *Can this tree be improved?*
- Relative fat-factor
  - *$rfat(T) \geq 0$*
  - Minimum tree is optimal
  - Overlaps and occupations are considered.
  - *Which of these trees is more optimal?*

Choosing Reference Points

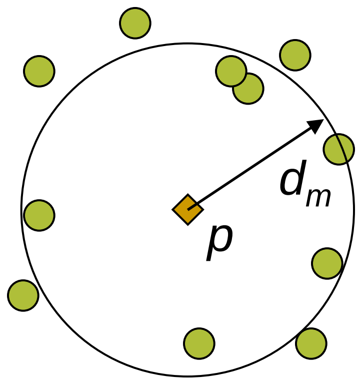
- All but naïve index structures need pivots (reference objects).
- Pivots are essential for partitioning and search pruning.
- Pivots influence performance:
  - Higher & more narrowly-focused distance density with respect to a pivot

$$\Downarrow$$

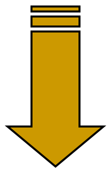
  - Greater change for a query object to be located at the most frequent distance to the pivot.

Choosing Reference Points (cont.)

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



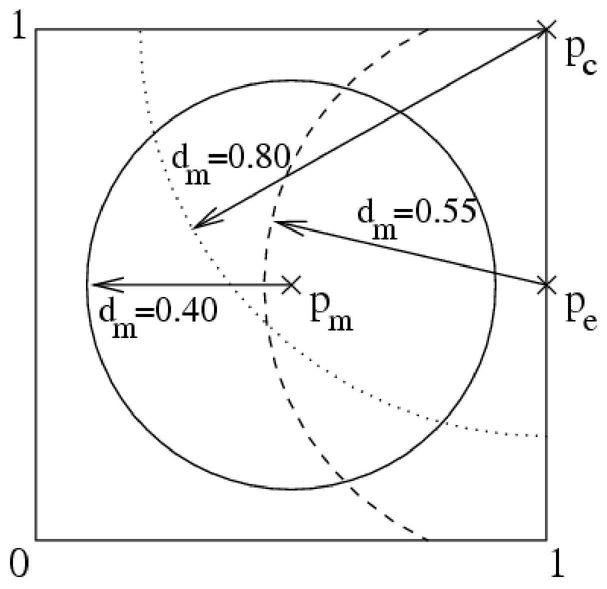
  - If all other distance are not very different



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

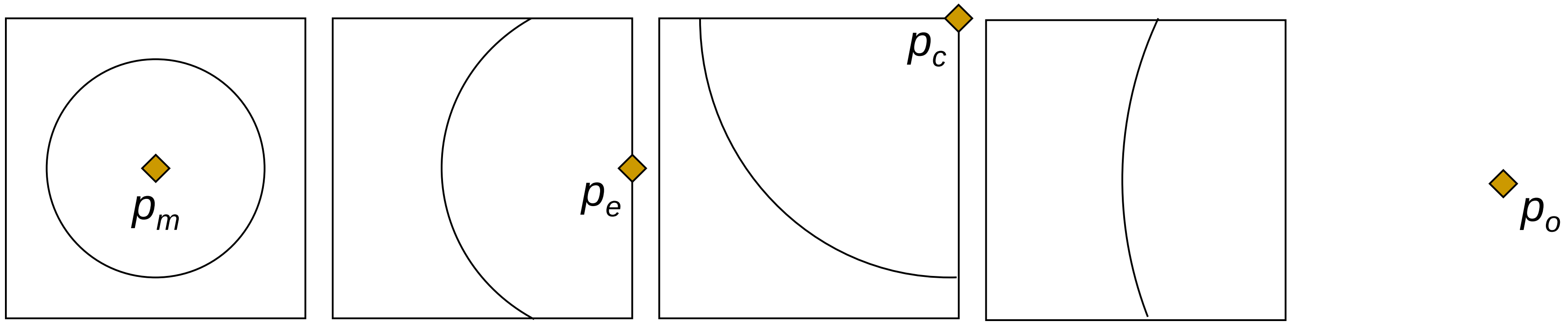
Choosing Reference Points: Example

- Position of a "good" pivot:
  - Unit square with uniform distribution
  - 3 positions: midpoint, edge, corner
  - Minimize the boundary length:
    - *len(p_m)=2.51*
    - *len(p_e)=1.256*
    - *len(p_c)=1.252*
- The best choice is at the border of space
- The midpoint is the worst alternative.
  - In clustering, the midpoint is the best.



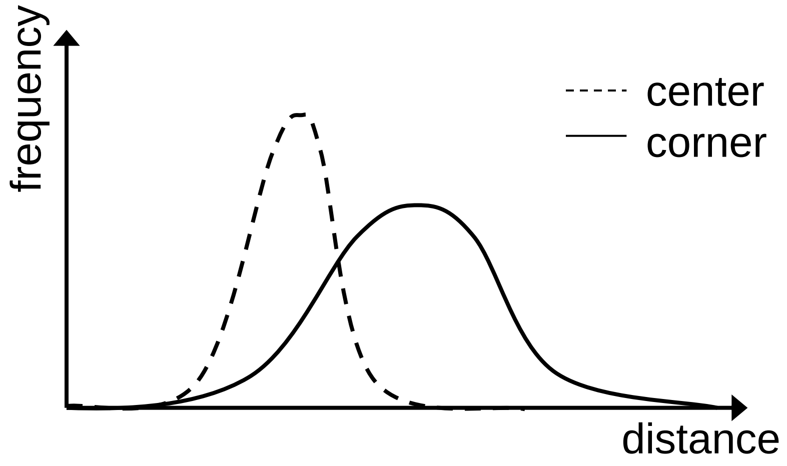
Choosing Reference Points: Example

- The shortest boundary has the pivot p_o outside the space.



Choosing Reference Points: Example

- Different view on a "good" pivot:
  - 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 Reference Points: Heuristics

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

- Algorithm for an outlier:
  1. Choose a random object
  2. Compute distances from this object to all others
  3. Pick the furthest object as 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:
  - 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. Minimum $min(d(p_1, p_3), d(p_2, p_3))$ is maximized.
  - …
  - Until we have *m* pivots.

Choosing More Pivots (cont.)

- This algorithm requires $O(3m \cdot m)$ distance computations.
  - For small values of *m*, it can be repeated several times for different candidate sets and
  - the best setting is used.

Choosing Pivots: Efficiency Criterion

- An algorithm based on *efficiency criterion:*
  - Measures ‘quality’ of sets of pivots.
  - Uses the mean distance $\mu_{\mathcal{D}}$ between pairs of objects in $\mathcal{D}$.

- Having two sets of pivots
  - $P_1=\{p_1, p_2, \ldots p_t\}$
  - $P_2=\{p'_1, p'_2, \ldots p'_t\}$
- P_1 is better than P_2 when $\mu_{\mathcal{D}P1} > \mu_{\mathcal{D}P2}$

Choosing Pivots: Efficiency Criterion

- Given a set of pivots $P=\{p_1, p_2, \ldots p_t\}$
- Estimation of $\mu_{\mathcal{D}^P}$ for P:

1. At random choose l pairs of objects $\{(o_1,o'_1), (o_2,o'_2), \ldots (o_l,o'_l)\}$ from database $X \subseteq \mathcal{D}$
2. Map all pairs into the feature space of the set of pivots P

   $\Psi(o_i)=(d(p_1,o_i), d(p_2,o_i), \ldots d(p_t,o_i))$

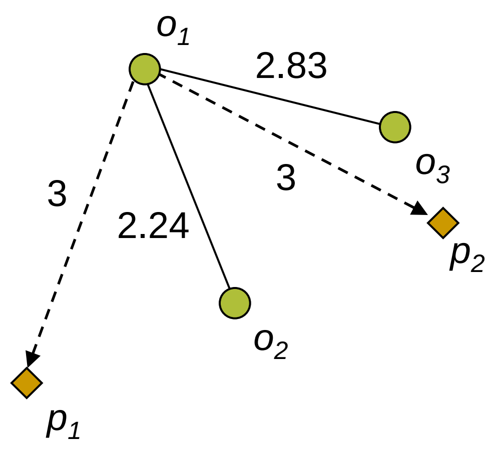
   $\Psi(o'_i)=(d(p_1,o'_i), d(p_2,o'_i), \ldots d(p_t,o'_i))$

1. For each pair (o_i,o'_i) compute their distance in the feature space: $d_i=L_\infty(\Psi(o_i), \Psi(o'_i))$.
2. Compute $\mu_{\mathcal{D}^P}$ as the mean of d_i : $\mu_{\mathcal{D}^P} = \frac{1}{l}\sum_{1\le i\le l} d_i$

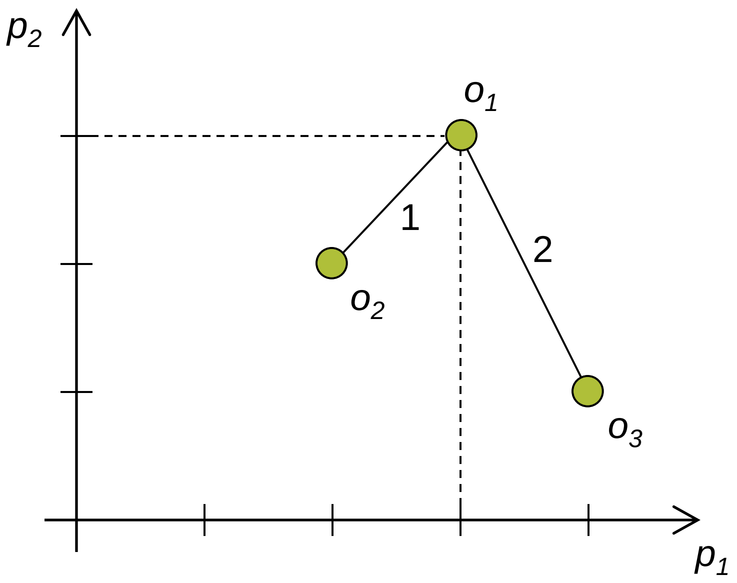
Efficiency Criterion: Example

- Having $P=\{p_1,p_2\}$
- Mapping used by $\mu_{\mathcal{D}^P}$:

Original space with d



Feature space with L_∞



Choosing Pivots: Incremental Selection

- Selects further pivots "on demand"
  - Based on efficiency criterion $\mu_{\mathcal{D}_P}$
- Algorithm:
  1. Select a sample set of *m* objects.
  2. $P_1=\{p_1\}$ is selected from the sample as $\mu_{\mathcal{D}_{P1}}$ is maximum.
  3. Select another sample set of *m* objects.
  4. Second pivot p_2 is selected as: $\mu_{\mathcal{D}_{P2}}$ is maximum where $P_2=\{p_1,p_2\}$ with p_1 fixed.
  5. …
- Total cost for selecting *k* pivots: *2lmk* distances
  - Next step would need *2lm* distance, if distances d_i for computing $\mu_{\mathcal{D}_P}$ are kept.

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.

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