SIMILARITY SEARCH
The Metric Space Approach

Pavel Zezula, Giuseppe Amato,
Vlastislav Dohnal, Michal Batko

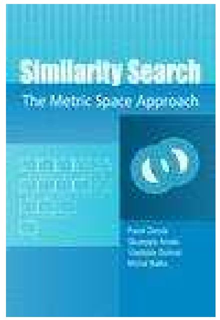


Table of Content

Survey of existing approaches

1. **ball partitioning methods**
2. generalized hyper-plane partitioning approaches
3. exploiting pre-computed distances
4. hybrid indexing approaches
5. approximated techniques

Survey of existing approaches

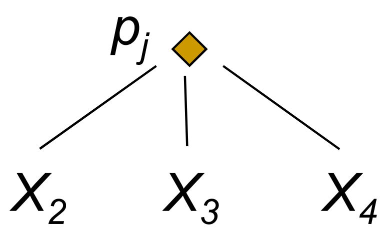
1. **ball partitioning methods**
   1. Burkhard-Keller Tree
   2. Fixed Queries Tree
   3. Fixed Queries Array
   4. Vantage Point Tree
      1. Multi-Way Vantage Point Tree
   5. Excluded Middle Vantage Point Forest
2. generalized hyper-plane partitioning approaches
3. exploiting pre-computed distances
4. hybrid indexing approaches
5. approximated techniques

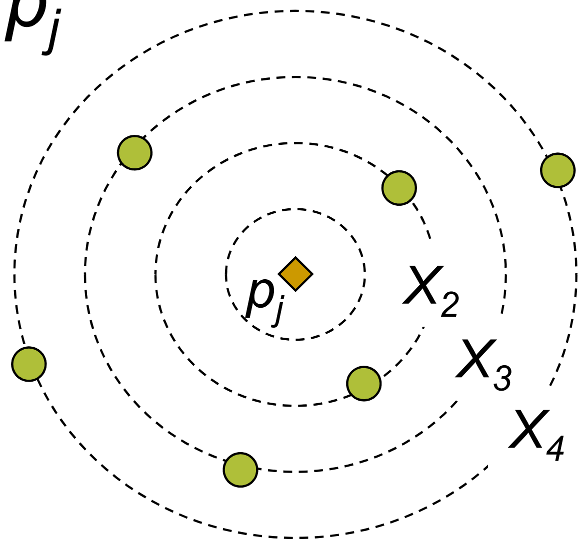
Burkhard-Keller Tree (BKT) [BK73]

- Applicable to discrete distance functions only
- Recursively divides a given dataset X
- Choose an arbitrary point $p_j \in X$, form subsets:

  $X_i = \{o \in X, d(o,p_j) = i\}$ for each distance $i \geq 0$.

- For each X_i create a sub-tree of p_j
  - empty subsets are ignored

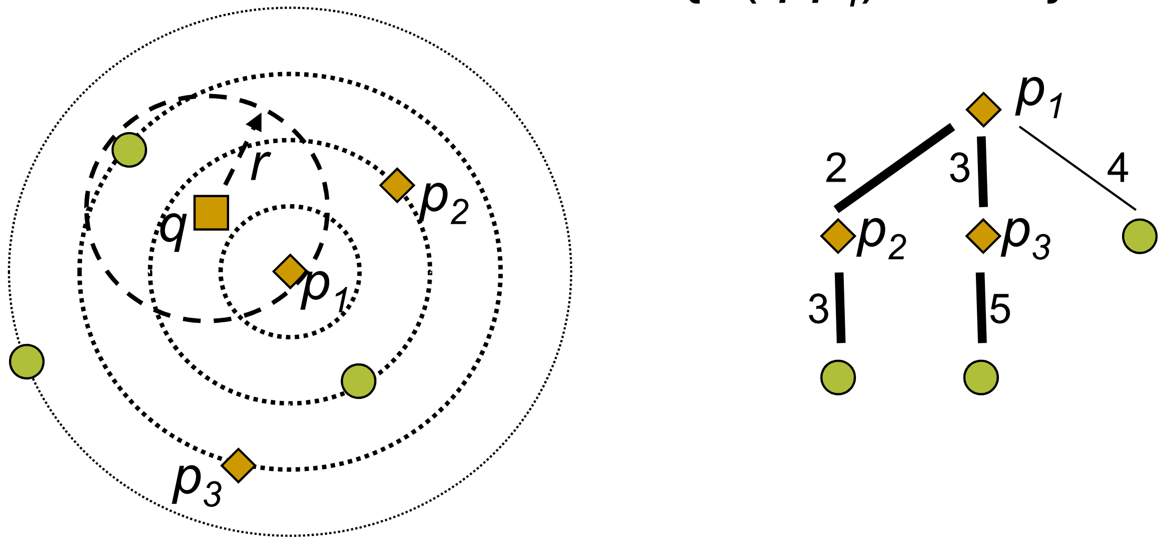




BKT: Range Query

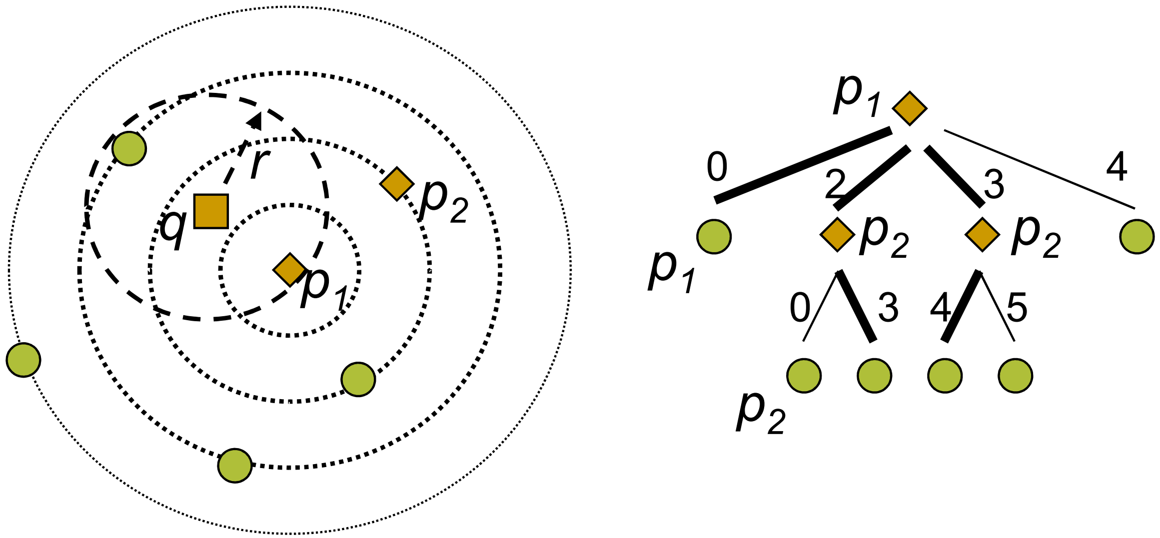
Given a query $R(q,r)$:

- 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 $\max\{d(q,p_i) - r, 0\} \leq i \leq d(q,p_i) + r$



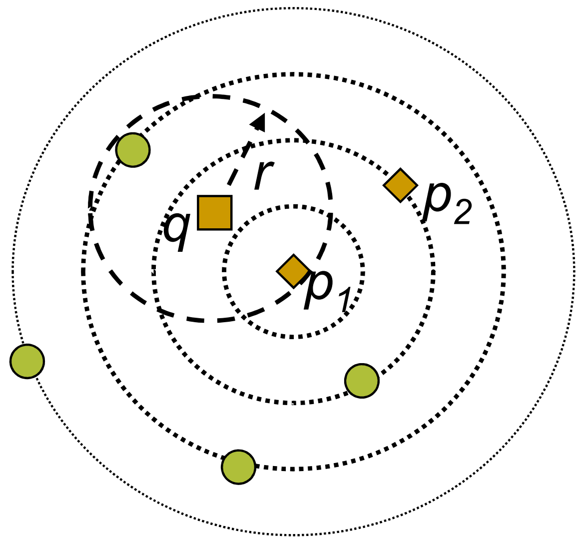
Fixed Queries Tree (FQT)

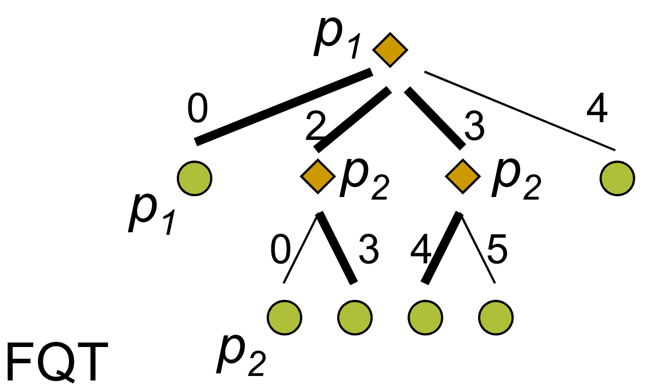
- modification of BKT
- each level has a single pivot
  - all objects stored in leaves
- during search distance computations are saved
  - usually more branches are accessed → one distance comp.

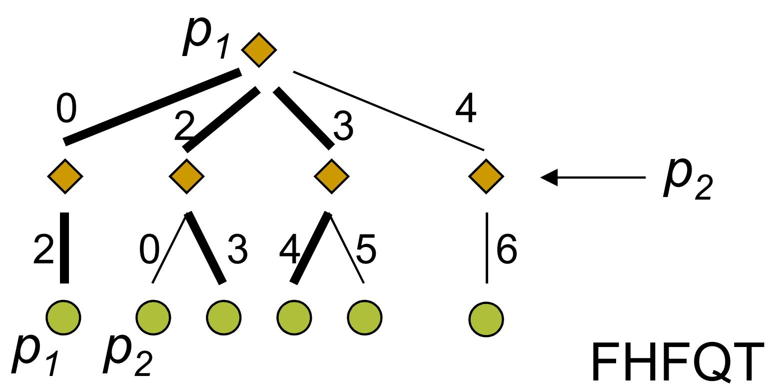


Fixed-Height FQT (FHFQT)

- extension of FQT
- all leaf nodes at the same level
  - increased filtering using more routing objects
  - extended tree depth does not typically introduce further computations

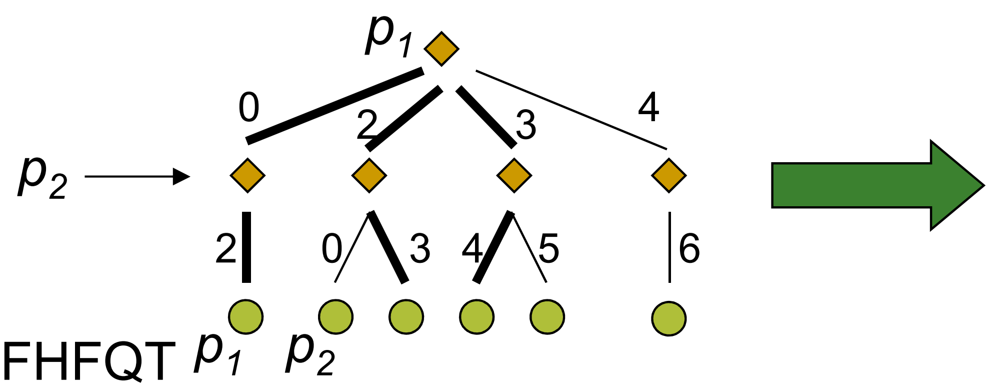






Fixed Queries Array (FQA)

- based on FHFQT
- an *h*-level tree is transformed to an array of paths
  - every leaf node is represented with a path from the root node
  - each path is encoded as *h* values of distance
- a search algorithm turns to a binary search in array intervals

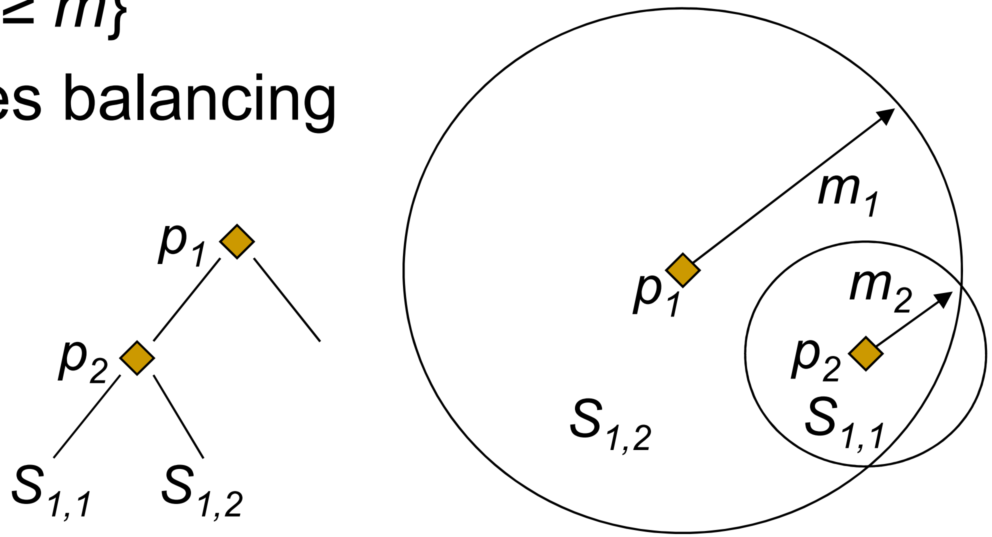


FHFQT

0	**2**	**2**	**3**	**3**	4	← p_1
2	0	**3**	**4**	5	6	← p_2

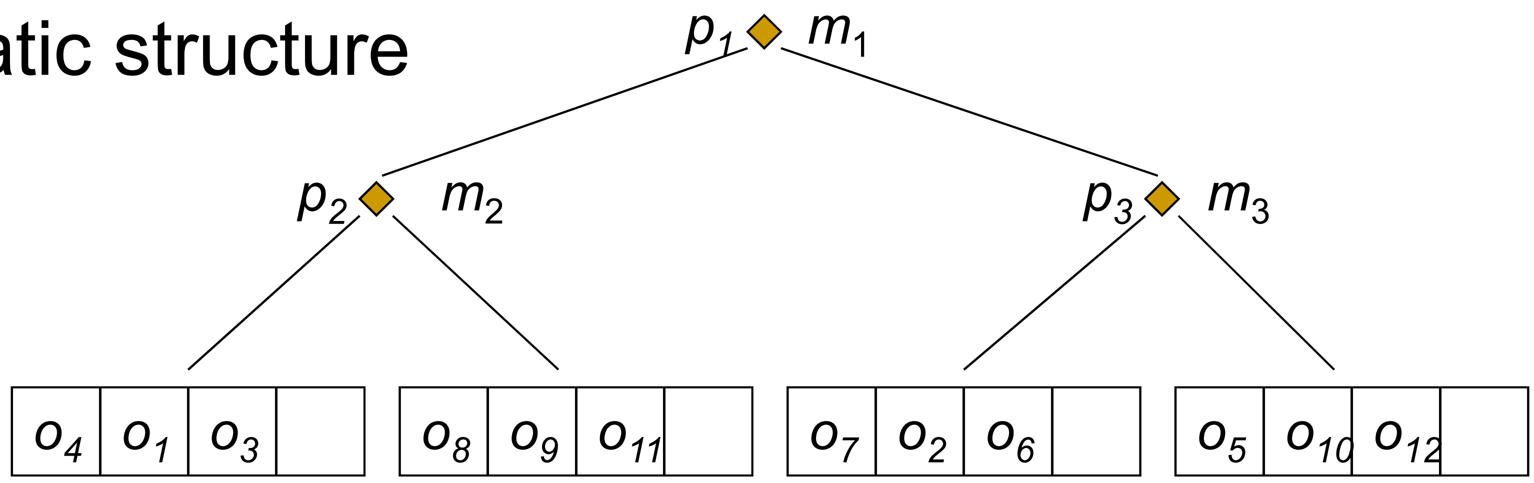
Vantage Point Tree (VPT)

- uses ball partitioning
  - recursively divides given data set X
- choose vantage point $p \in X$, compute 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



VPT (cont.)

- One or more objects can be accommodated in leaves.
- VP tree is a balanced binary tree.
- Static structure

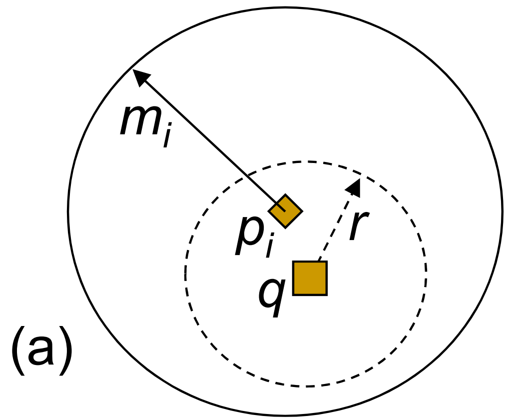


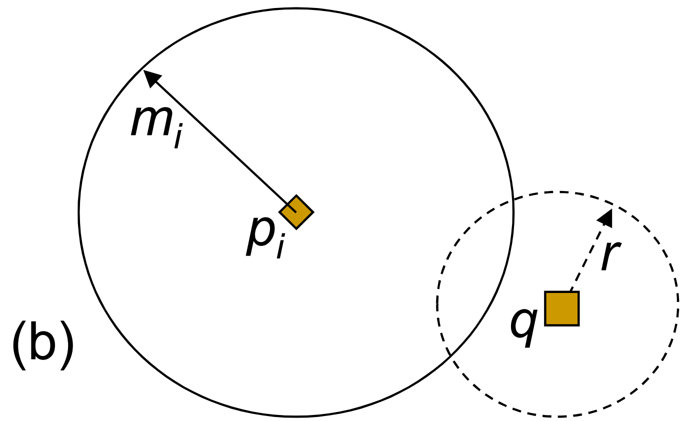
- Pivots p_1,p_2 and p_3 belong to the database!
- In the following, we assume just one object in a leaf.

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)





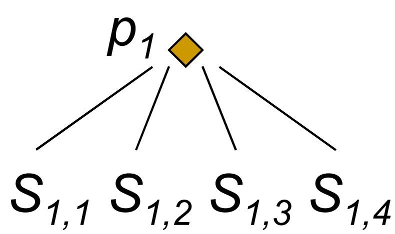
VPT: *k*-*NN* Search

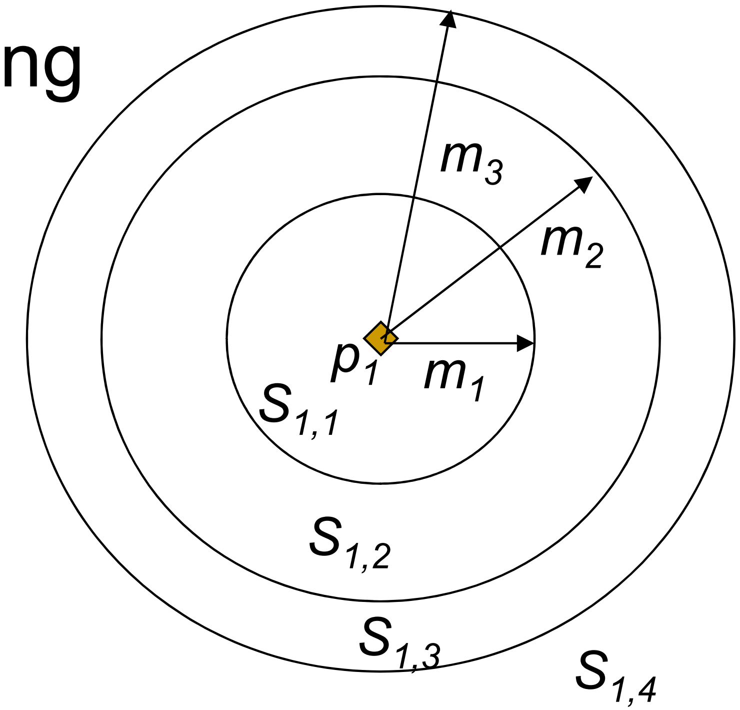
Given a query $NN(q)$:

- 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

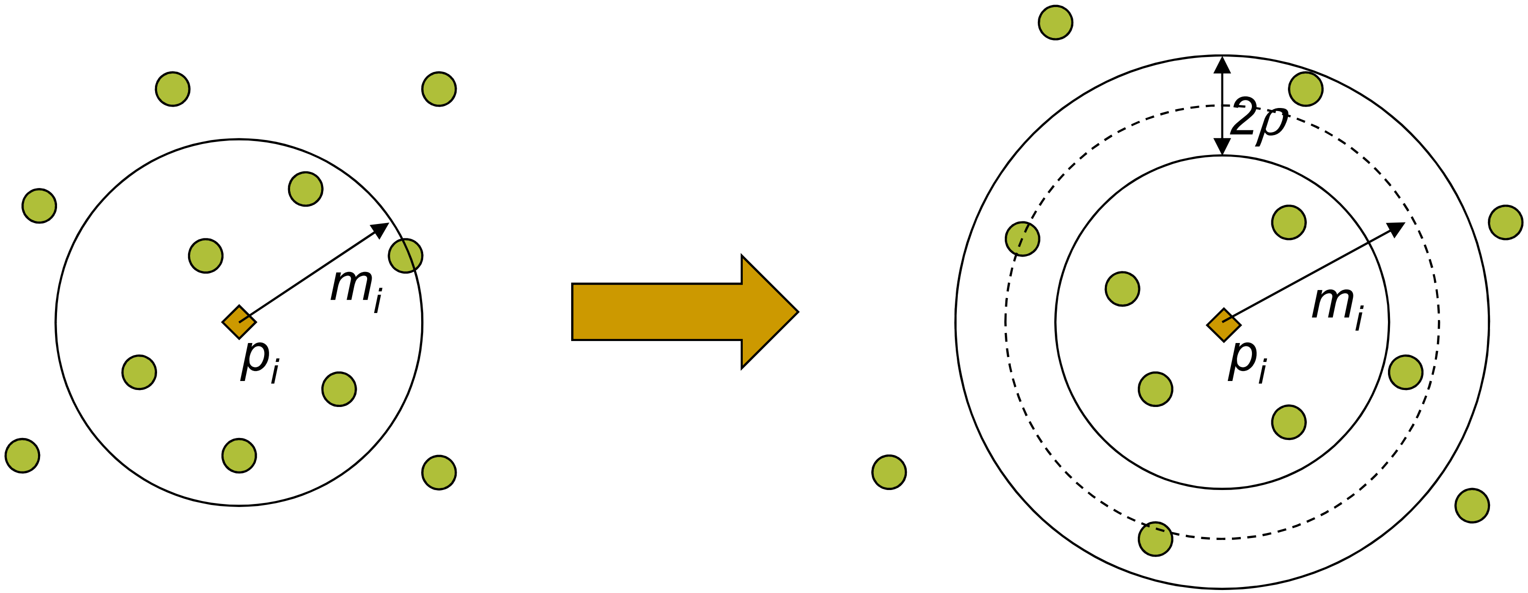
- inherits all principles from VPT
  - but partitioning is modified
- *m*-ary balanced tree
- applies multi-way ball partitioning





Vantage Point Forest (VPF)

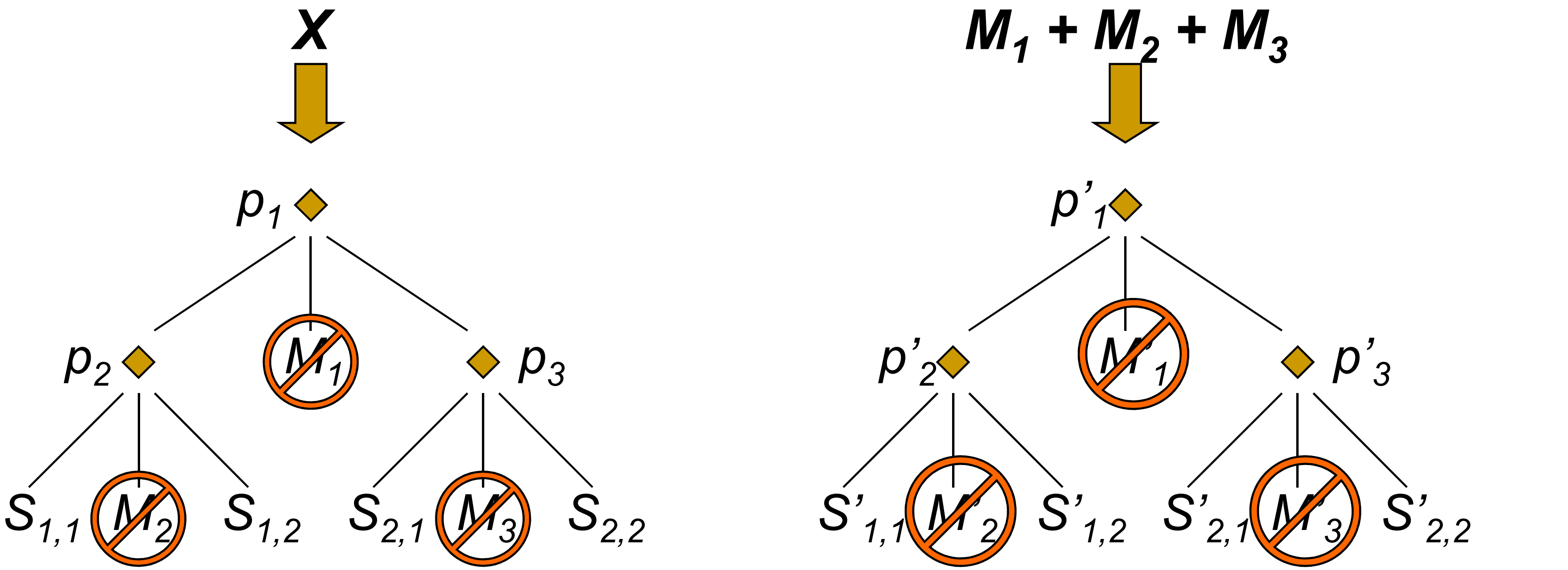
- a forest of binary trees
- uses excluded middle partitioning



- middle area is excluded from the process of tree building

VPF (cont.)

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



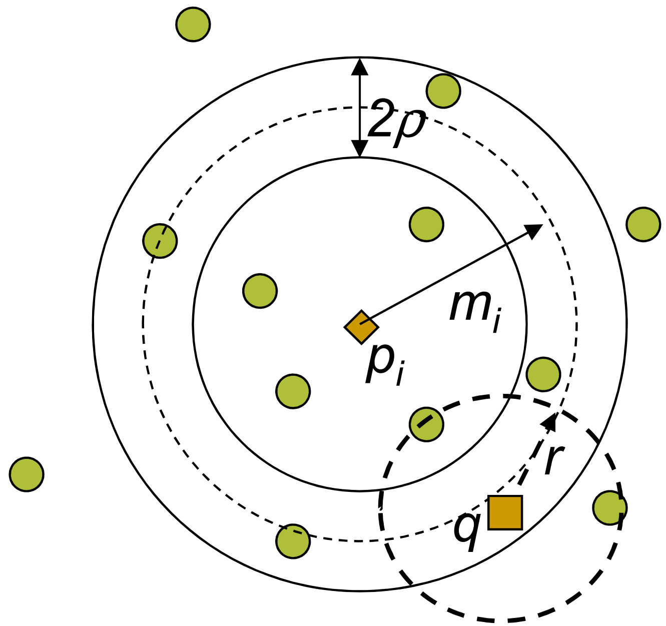
VPF: Range Search

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

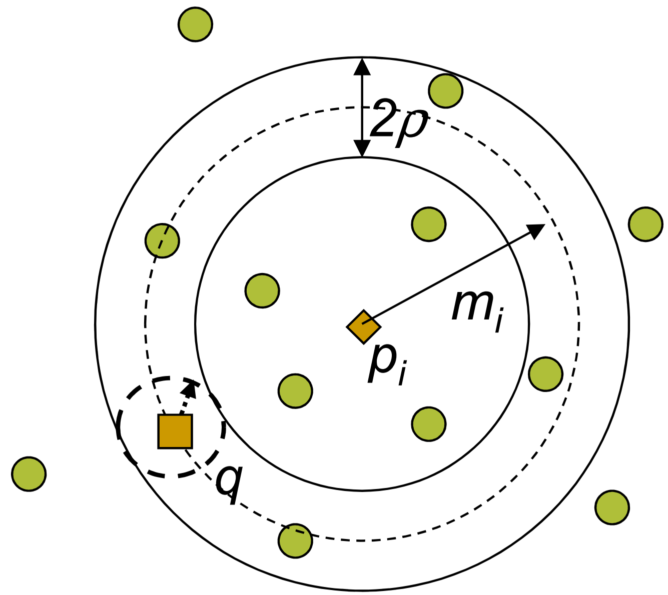
- start with the first tree
  - traverse the tree starting from its root
  - in each internal node (p_i,m_i), do:
    - if $d(q,p_i) \leq r$ — report p_i
    - if $d(q,p_i) - r \leq m_i - \rho$ — search the left sub-tree
      - if $d(q,p_i) + r \geq m_i - \rho$ — search the next tree !!!
    - if $d(q,p_i) + r \geq m_i + \rho$ — search the right sub-tree
      - if $d(q,p_i) - r \leq m_i + \rho$ — search the next tree !!!
    - if $d(q,p_i) - r \geq m_i - \rho$ and $d(q,p_i) + r \leq m_i + \rho$ — search only the next tree !!!

VPF: Range Search (cont.)

- Query intersects all partitions
  - Search both sub-trees
  - Search the next tree



- Query collides only with exclusion
  - Search just the next tree



Survey of existing approaches

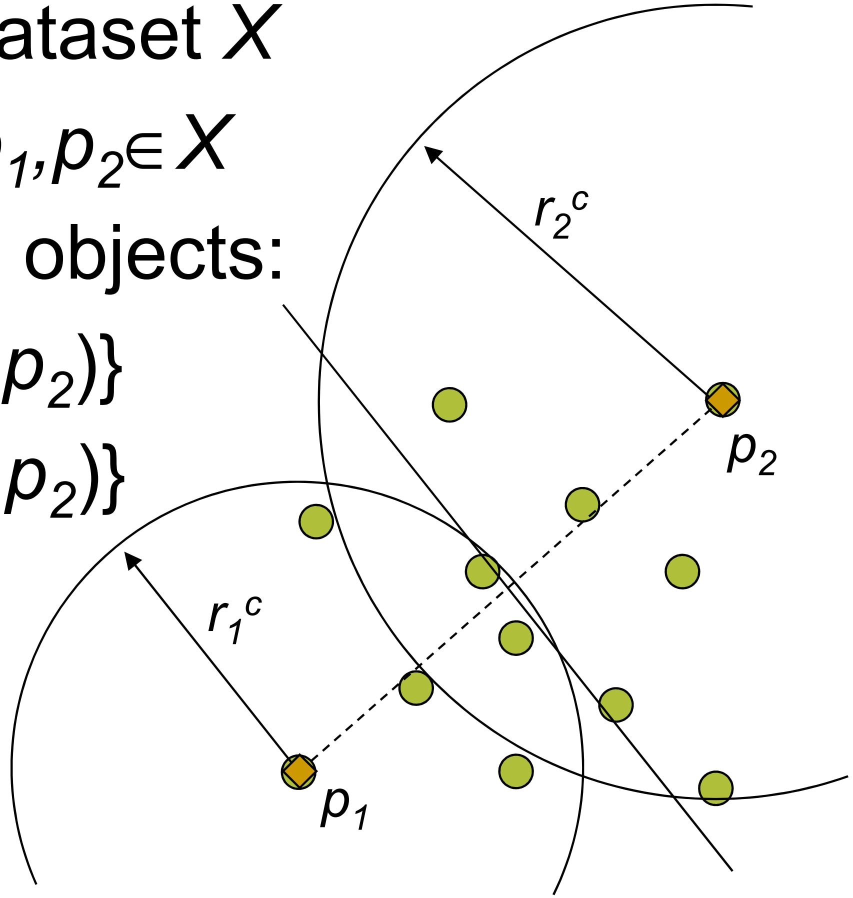
1. ball partitioning methods
2. **generalized hyper-plane partitioning approaches**
    1. Bisector Tree
    2. Generalized Hyper-plane Tree
3. exploiting pre-computed distances
4. hybrid indexing approaches
5. approximated techniques

Bisector Tree (BT)

- Applies generalized hyper-plane partitioning
- Recursively divides a given dataset X
- Choose two arbitrary points $p_1, p_2 \in X$
- Form subsets from remaining objects:

  $$S_1 = \{o \in X, d(o,p_1) \leq d(o,p_2)\}$$
  $$S_2 = \{o \in X, d(o,p_1) > d(o,p_2)\}$$

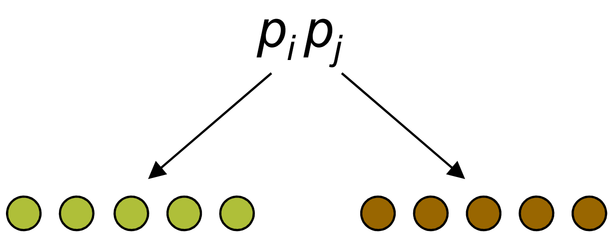
- Covering radii r_1^c and r_2^c are established:
  - The balls can intersect!

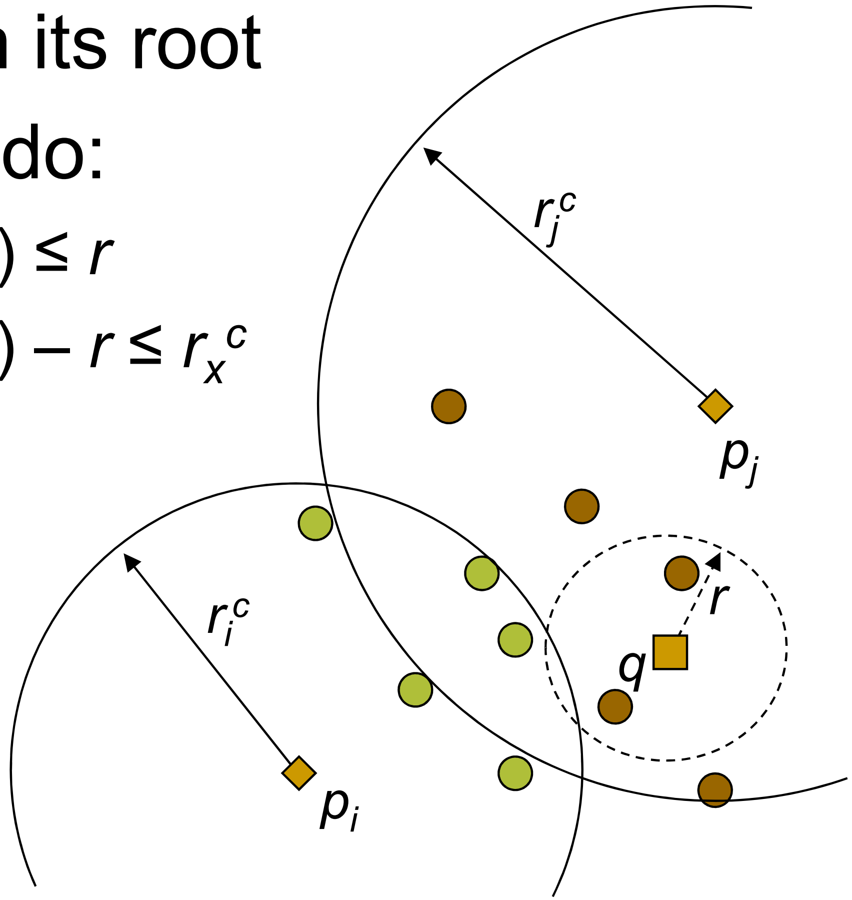


BT: Range Query

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

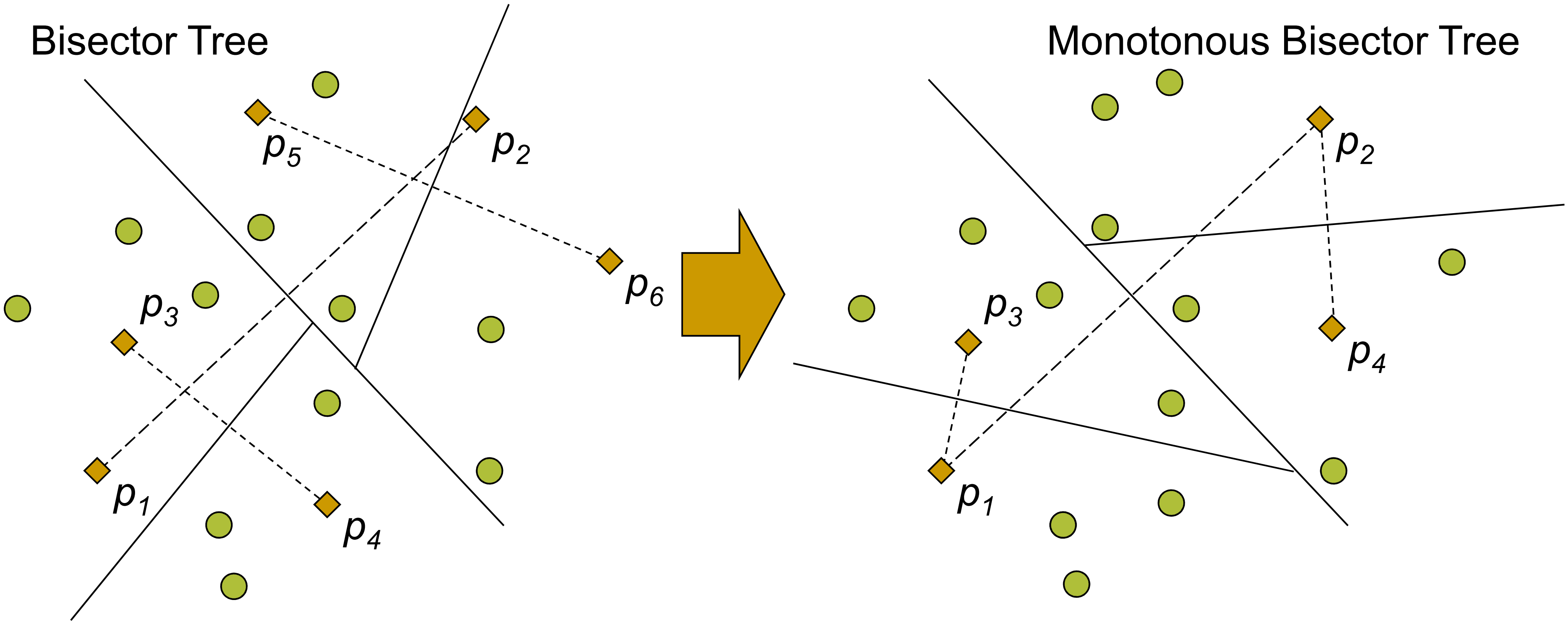
- traverse the tree starting from its root
- in each internal node $\langle p_i,p_j \rangle$, do:
  - report p_x on output if $d(q,p_x) \le r$
  - enter a child of p_x if $d(q,p_x) - r \le r_x^c$





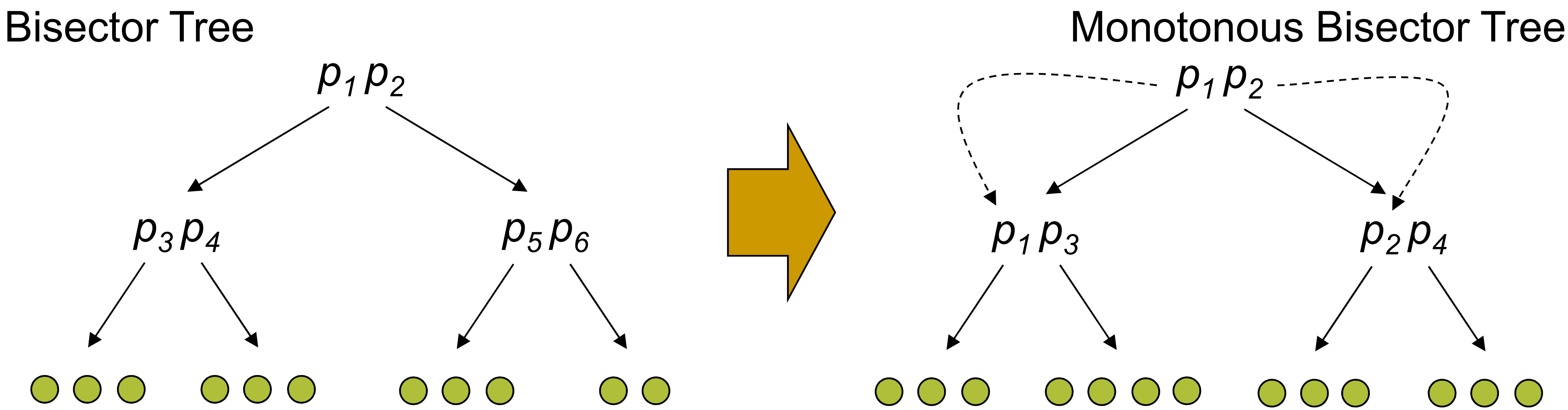
Monotonous Bisector Tree (MBT)

- A variant of Bisector Tree
- Child nodes inherit one pivot from the parent.
  - For convenience, no covering radii are shown.



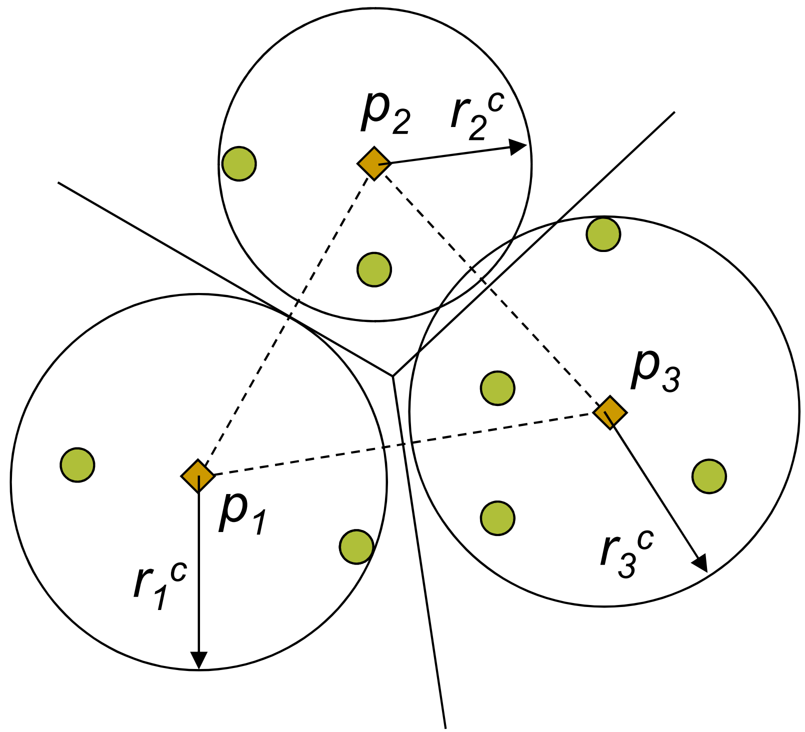
MBT (cont.)

- Fewer pivots used → fewer distance evaluations during query processing & more objects in leaves.



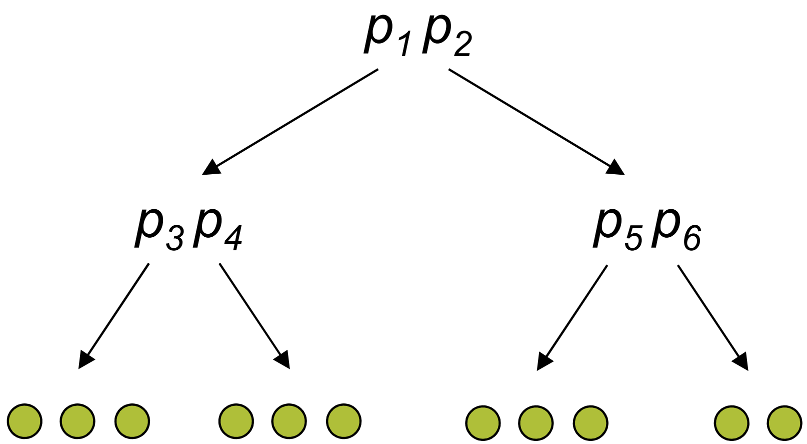
Voronoi Tree

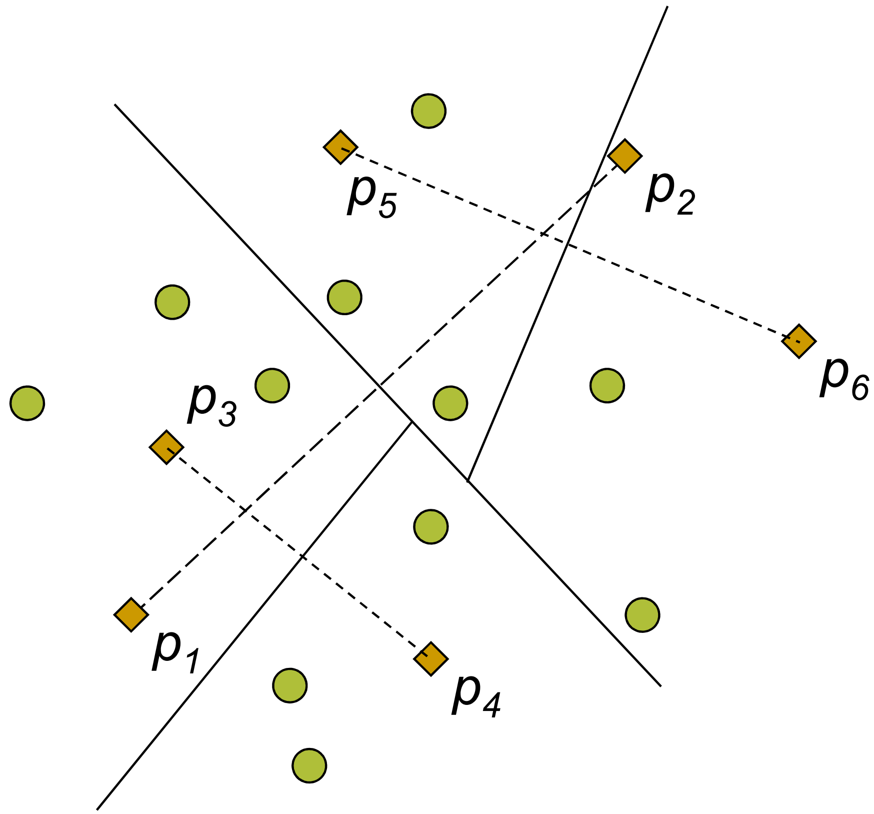
- Extension of Bisector Tree
- Uses more pivots in each internal node
  - Usually three pivots



Generalized Hyper-plane Tree (GHT)

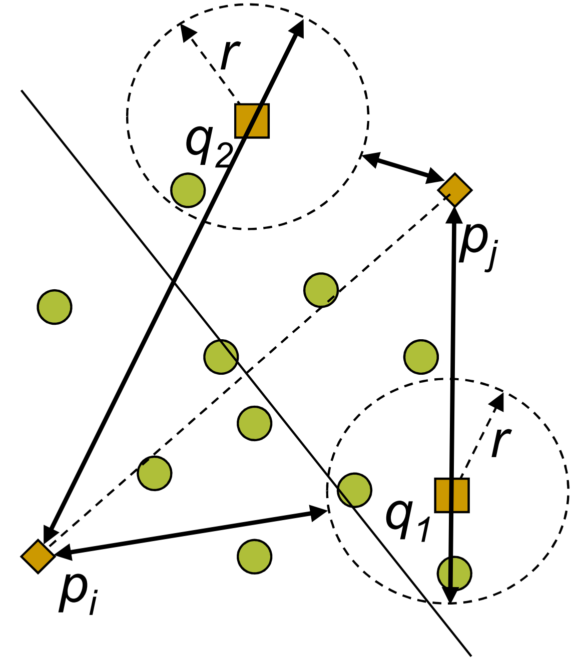
- Similar to Bisector Trees
- Covering radii are not used





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 <p_i,p_j>, 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 $d(q,p_i) + r \geq d(q,p_j) - r$

Survey of existing approaches

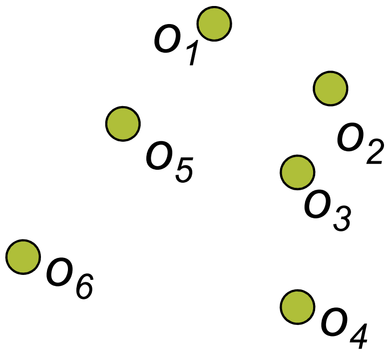
1. ball partitioning methods
2. generalized hyper-plane partitioning approaches
3. **exploiting pre-computed distances**
    1. AESA
    2. Linear AESA
    3. Other Methods – Shapiro, Spaghettis
4. hybrid indexing approaches
5. approximated techniques

Exploiting Pre-computed Distances

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

AESA

- Approximating and Eliminating Search Algorithm
- Matrix *n*×*n* of distances is stored
  - 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

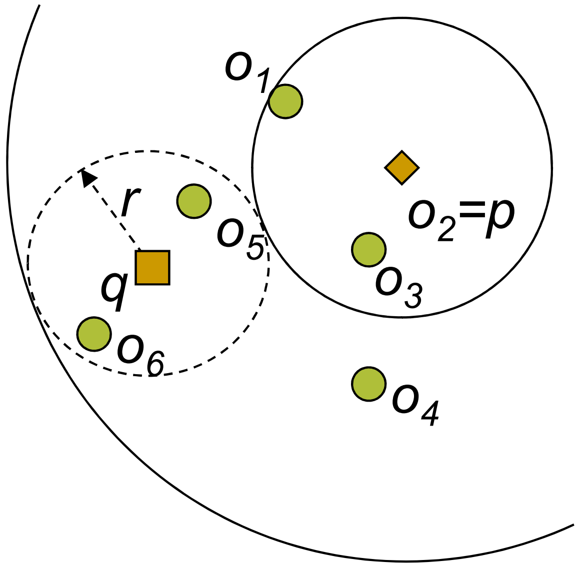
- Every object can play a role of *pivot.*

AESA: Range Query

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

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

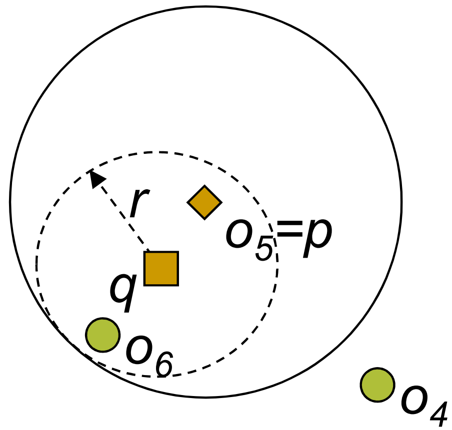
	~~o_1~~	~~$\mathbf{o_2}$~~	~~o_3~~	o_4	o_5	o_6
~~o_1~~		**1.6**	2.0	3.5	1.6	3.6
~~$\mathbf{o_2}$~~			**1.0**	**2.6**	**2.6**	**4.2**
~~o_3~~				1.6	2.1	3.5
o_4					3.0	3.4
o_5						2.0
o_6						



AESA: Range Query (cont.)

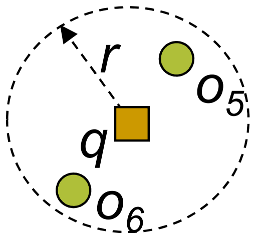
- From remaining objects, select another object as pivot *p*.
  - To maximize pruning, select the closest object to *q*.
  - It maximizes the lower bound on distances $|d(q,p) - d(p,o)|$.
- Filter out objects using *p*.

	o_1	o_2	o_3	~~o_4~~	$\mathbf{o_5}$	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
$\mathbf{o_5}$						**2.0**
o_6						



AESA: Range Query (cont.)

- This process is repeated until the number of remaining objects is small enough
  - Or all objects have been used as pivots.
- Check remaining objects directly with *q*.
  - Report *o* if $d(q,o) \leq r$.

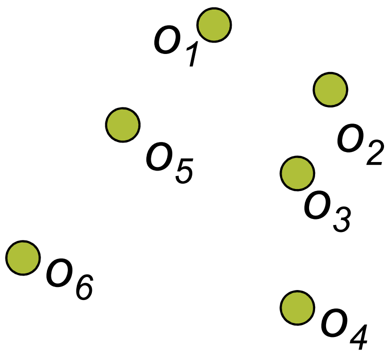


	o_1	o_2	o_3	o_4	o_5	o_6
o_1		1.6	2.0	3.5	1.6	3.6
o_2			1.0	2.6	2.6	4.2
o_3				1.6	2.1	3.5
o_4					3.0	3.4
o_5						2.0
o_6						

- Objects *o* that fulfill $d(q,p)+d(p,o) \leq r$ can directly be reported on the output without further checking.
  - E.g. o_5, because it was the pivot in the previous step.

Linear AESA (LAESA)

- 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.

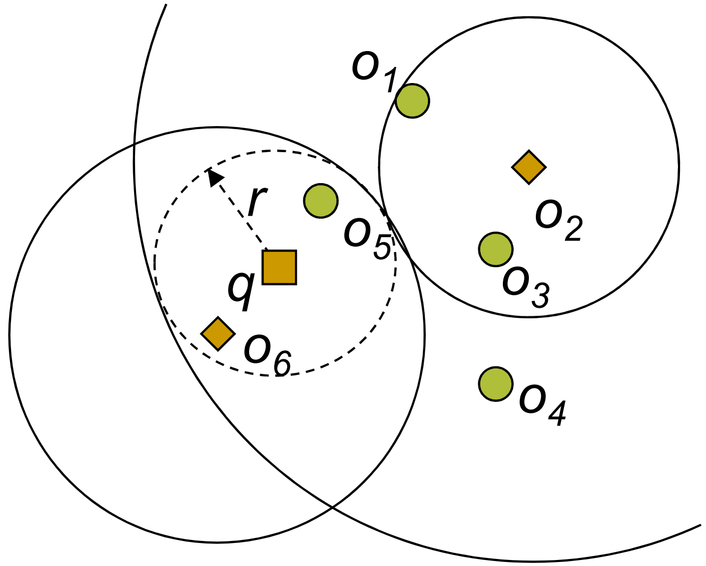


		o_1	o_2	o_3	o_4	o_5	o_6
pivots	o_2	1.6	0	1.0	2.6	2.6	4.2
	o_6	3.6	4.2	3.5	3.4	2.0	0

LAESA: Range Query

- Due to limited number of pivots, the algorithm differs.
- We need not be able to select a pivot among non-discarded objects.
  - First, all pivots are used for filtering.
  - Next, remaining objects are directly compared to *q*.

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

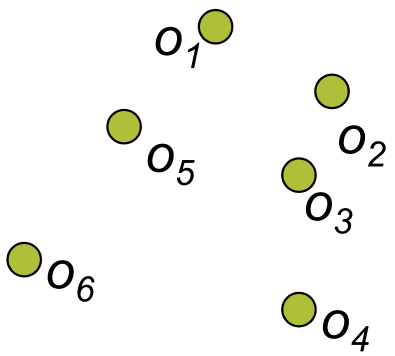


LAESA: Summary

- AESA and LAESA tend to be linear in distance computations
  - For larger query radii or higher values of *k*

Shapiro's LAESA

- Very similar to LAESA
- Database objects are sorted with respect to the first pivot.



pivots		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

Shapiro's LAESA: Range Query

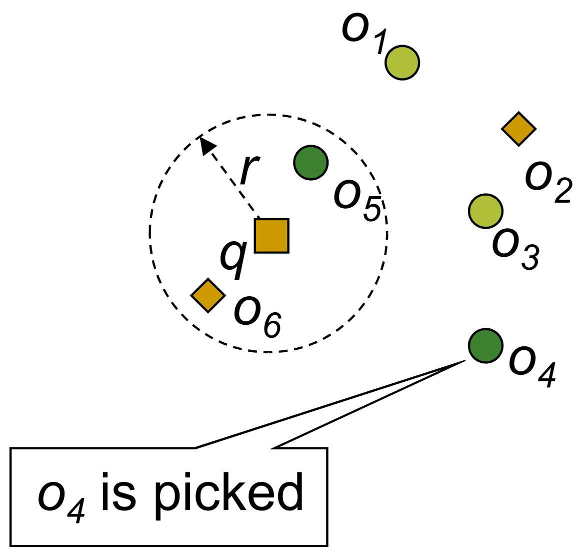
Given a query $R(q,r)$:

- Compute $d(q,p_1)$
- Start with object o_i "closest" to q
  - i.e. $|d(q,p_1) - d(p_1,o_i)|$ is minimal

$p_1 = o_2$

$d(q,o_2) = 3.2$

	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



Shapiro's LAESA: Range Query (cont.)

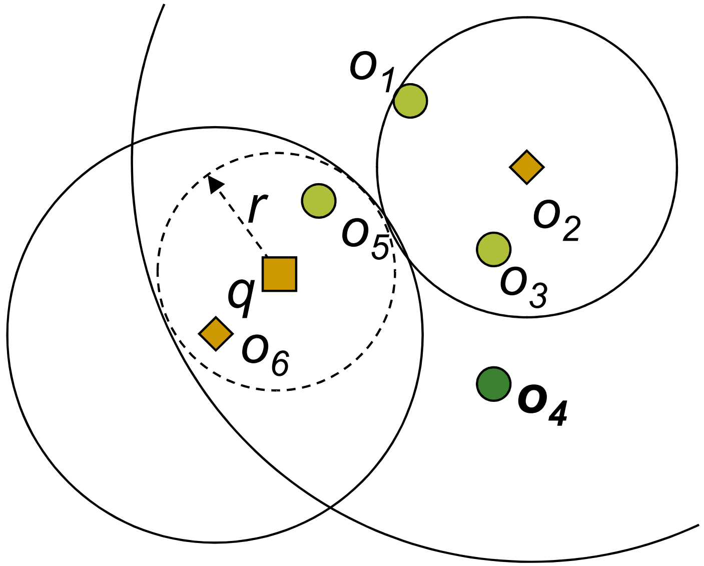
- Next, o_i is checked against all pivots
  - Discard it if $|d(q,p_j) - d(p_j,o_i)| > r$ for any p_j
  - If not eliminated, check $d(q,o_i) \leq r$

$R(q,1.4)$

$d(q,o_2) = 3.2$

$d(q,o_6) = 1.2$

	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



Shapiro's LAESA: Range Query (cont.)

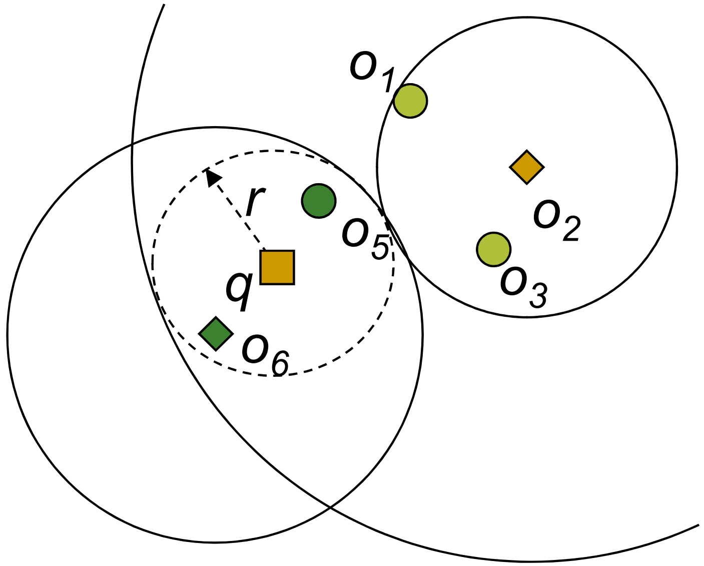
- Search continues with objects o_{i+1}, o_{i-1}, o_{i+2}, o_{i-2}, …
  - Until conditions $|d(q,p_1) - d(p_1,o_{i+?})| > r$ and $|d(q,p_1) - d(p_1,o_{i-?})| > r$ hold

$p_1 = o_2$ $\qquad d(q,o_2) = 3.2$

	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

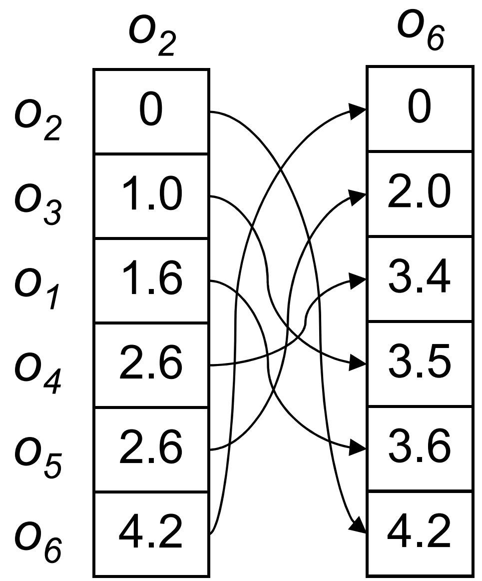
$|d(q,o_2) - d(o_2,o_1)| = 1.6 > 1.4$

$|d(q,o_2) - d(o_2,o_6)| = 1 \leq 1.4$



Spaghettis

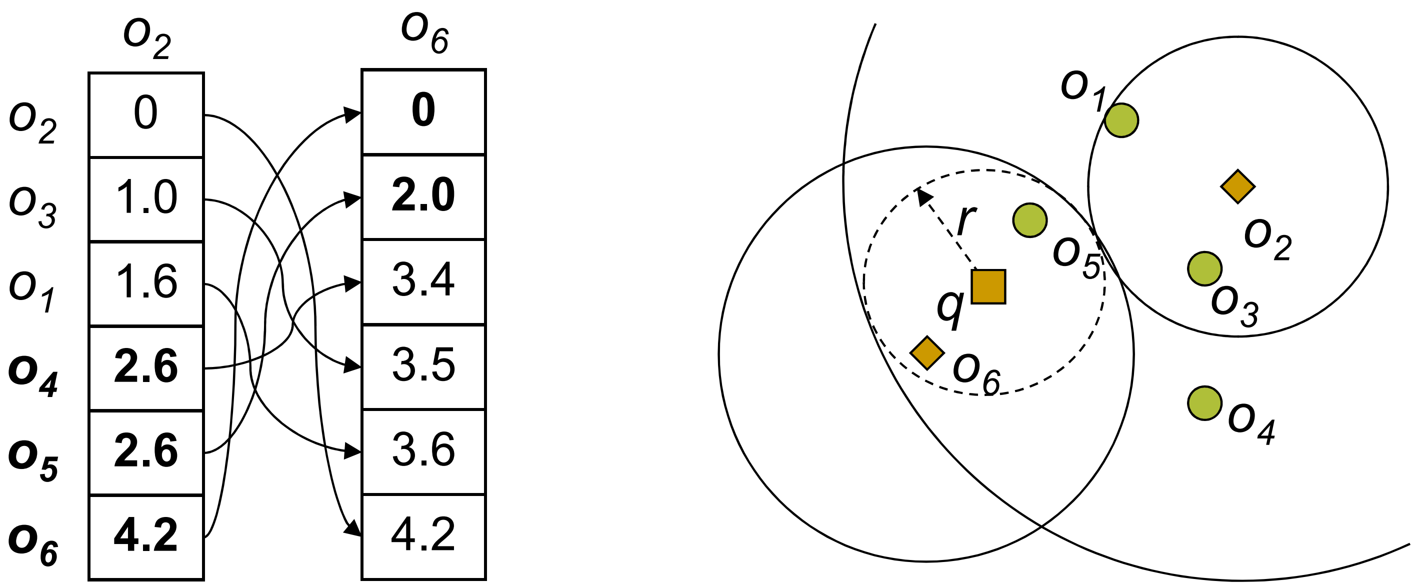
- Improvement of LAESA
- Matrix $m \times n$ is stored in m arrays of length n.
- Each array is sorted according to the distances in it.
- Position of object o can vary from array to array
  - Pointers (or array permutations) with respect to the preceding array must be stored.



Spaghettis: Range Query

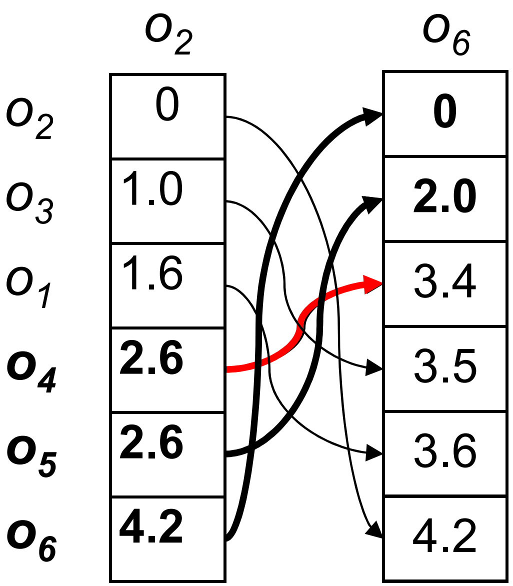
Given a query $R(q,r)$:

- Compute distances to pivots, i.e. $d(q,p_i)$
- One interval is defined on each of m arrays
  - $[\, d(q,p_i) - r,\ d(q,p_i) + r \,]$ for all $1 \le i \le m$

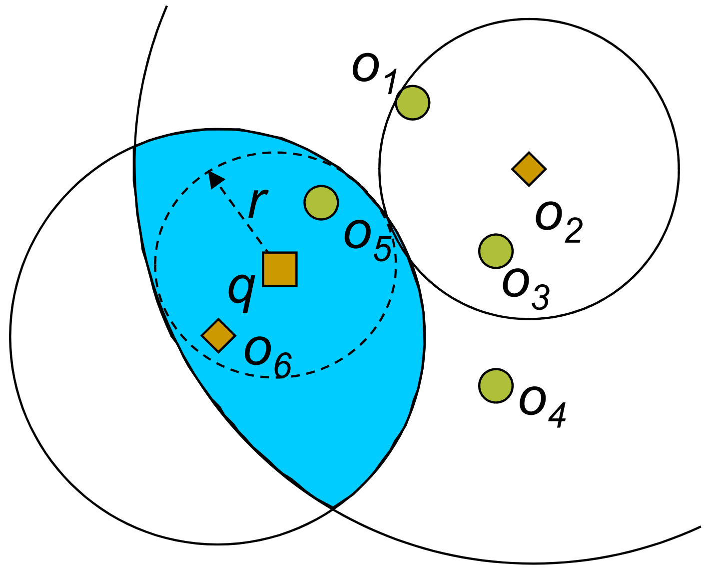


Spaghettis: Range Query (cont.)

- Qualifying objects lie in the intervals' intersection.
  - Pointers are followed from array to array.
- Non-discarded objects are checked against *q*.



Response: o_5, o_6



Survey of existing approaches

1. ball partitioning methods
2. generalized hyper-plane partitioning approaches
3. exploiting pre-computed distances
4. **hybrid indexing approaches**
    1. Multi Vantage Point Tree
    2. Geometric Near-neighbor Access Tree
    3. Spatial Approximation Tree
    4. M-tree
    5. Similarity Hashing
5. approximated techniques

Introduction

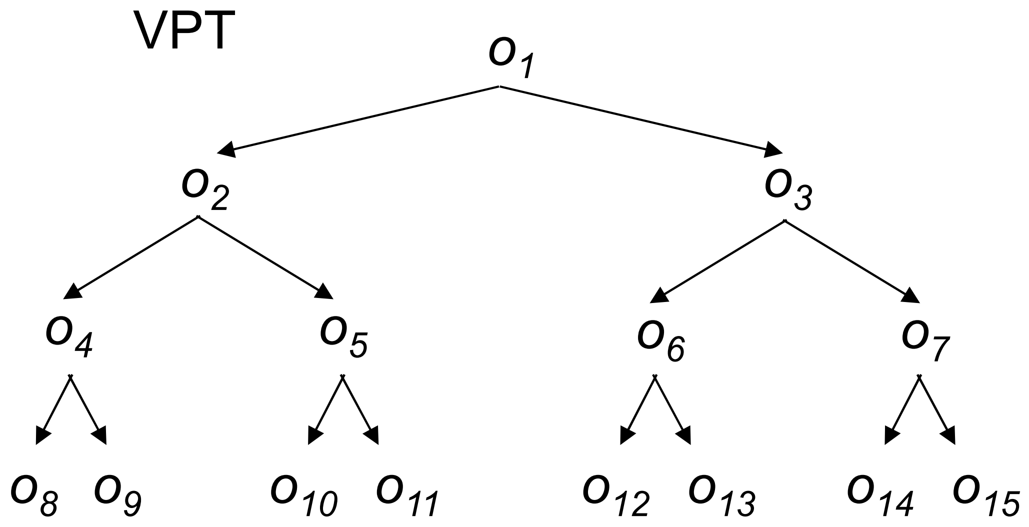
- Structures that store pre-computed distances have high space requirements
  - But good performance boost during query processing.
- Hybrid approaches combine partitioning and pre-computed distances into a single system
  - Less space requirements
  - Good query performance

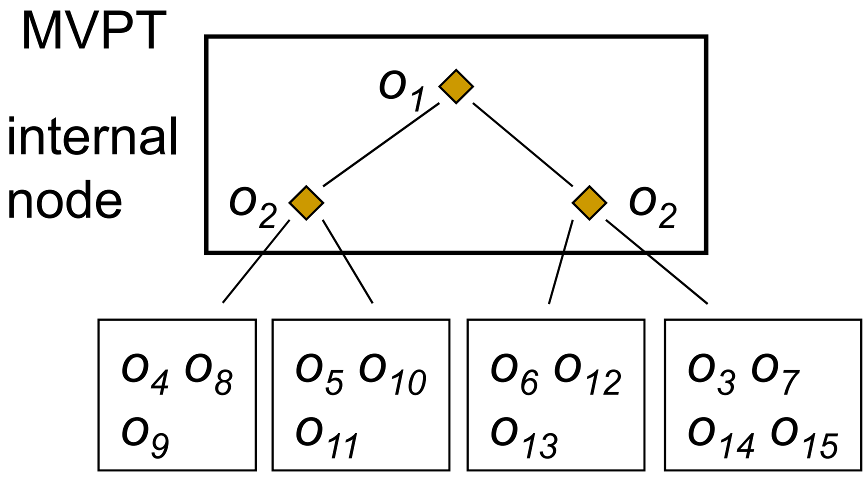
Multi Vantage Point Tree (MVPT)

- Based on Vantage Point Tree (VPT)
  - Targeted to static collections as well.
- Tries to decrease the number of pivots
  - With the aim of improving performance in terms of distance computations.
- Stores distances to pivots in leaves
  - These distances are evaluated during insertion of objects.
- No object duplication
  - Objects playing the role of a pivot are stored only in internal nodes.
- Leaf nodes can contain more than one object.

MVPT: Structure

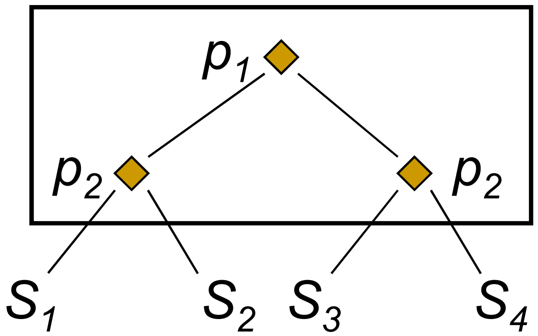
- Two pivots are used in each internal node
  - VPT uses just one pivot.
  - Idea: two levels of VPT collapsed into a single node

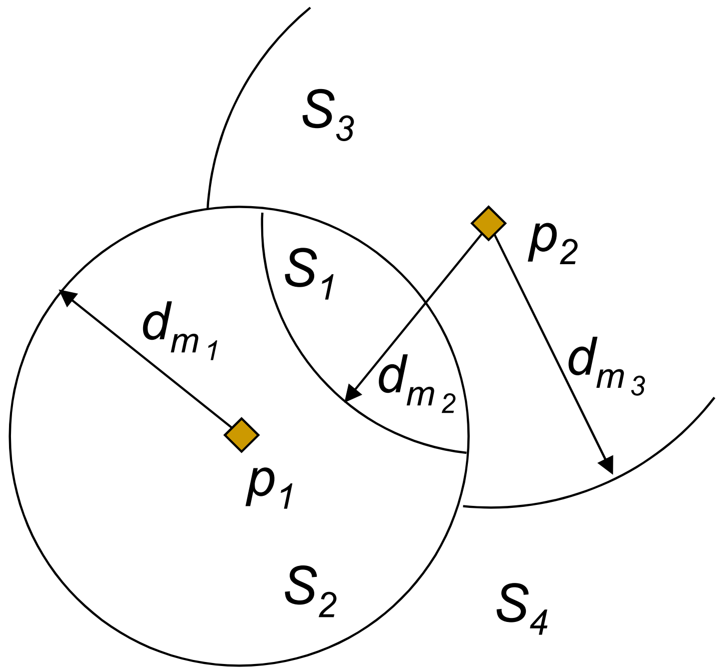




MPVT: Internal Node

- Ball partitioning is applied
  - Pivot p_2 is shared

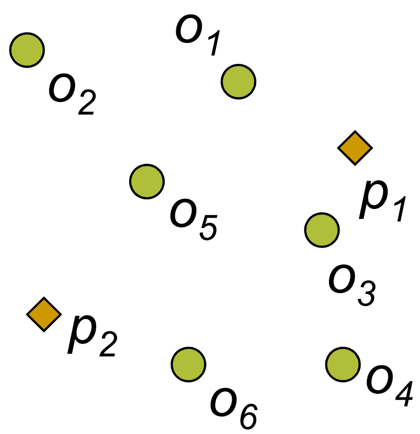




- In general, MVPT can use *k* pivots in a node
  - Number of children is 2^k !!!
  - Multi-way partitioning can be used as well $\rightarrow m^k$ children

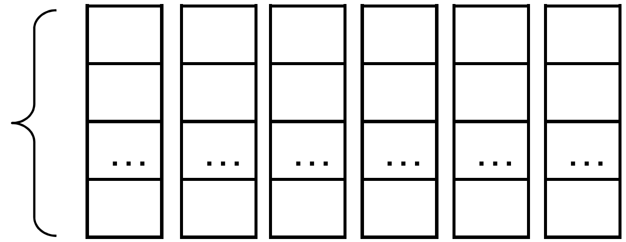
MVPT: Leaf Node

- 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.



	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

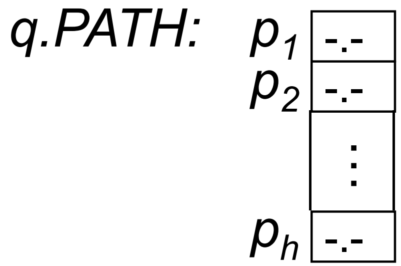
Distances from objects to the first *h* pivots on the path from the root



MVPT: Range Search

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

- Initialize the array *PATH* of *h* distances from *q* to the first *h* pivots.
  - Values are initialized to undefined.



- Start in the root node and traverse the tree (depth-first).

MVPT: Range Search (cont.)

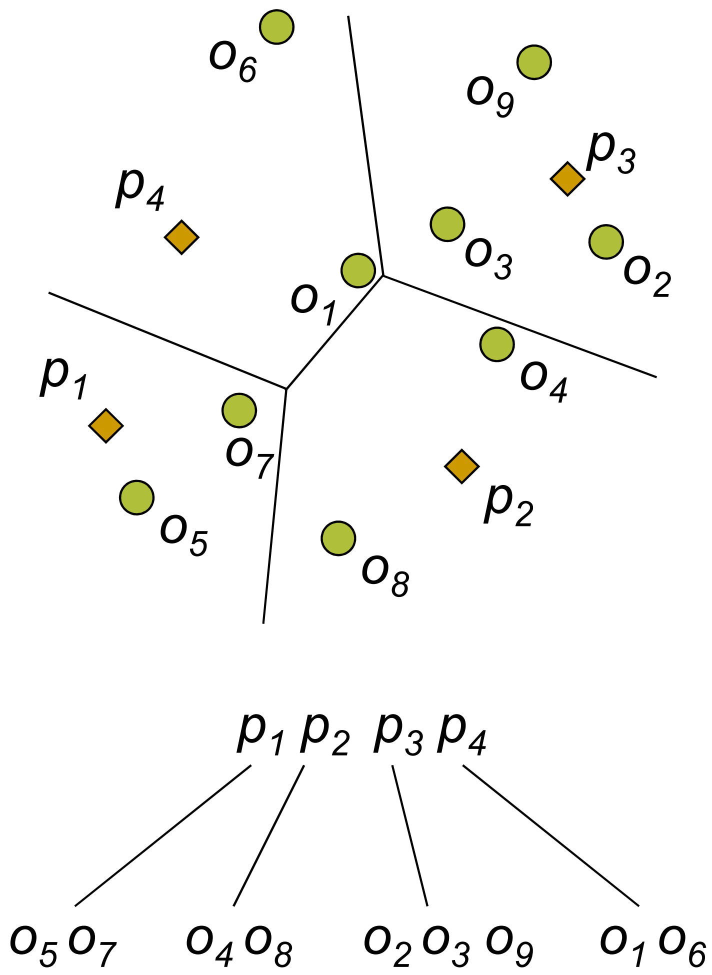
- In an internal node with pivots p_i , p_{i+1}:
- Compute distances $d(q,p_i)$, $d(q,p_{i+1})$
  - Store in *q.PATH*
    - if they are within the first *h* pivots from the root.
  - If $d(q,p_i) \le r$ output p_i
  - If $d(q,p_{i+1}) \le r$ output p_{i+1}
  - If $d(q,p_i) \le d_{m1}$
    - If $d(q,p_{i+1}) \le d_{m2}$ visit the first branch
    - If $d(q,p_{i+1}) \ge d_{m2}$ visit the second branch
  - If $d(q,p_i) \ge d_{m1}$
    - If $d(q,p_{i+1}) \le d_{m3}$ visit the third branch
    - If $d(q,p_{i+1}) \ge d_{m3}$ visit the fourth branch

MVPT: Range Search (cont.)

- In a leaf node with pivots p_1, p_2 and objects o_i:
- Compute distances $d(q,p_1)$, $d(q,p_2)$
  - If $d(q,p_i) \leq r$ output p_i
  - If $d(q,p_{i+1}) \leq r$ output p_{i+1}
- For all objects $o_1,\ldots,o_c$:
  - If $d(q,p_1) - r \leq d(o_i,p_1) \leq d(q,p_1) + r$ and
    $d(q,p_2) - r \leq d(o_i,p_2) \leq d(q,p_2) + r$ and
    $\forall p_j$: $q.PATH[j] - r \leq o_i.PATH[j] \leq q.PATH[j] + r$
    - Compute $d(q,o_i)$
    - If $d(q,o_i) \leq r$ output o_i

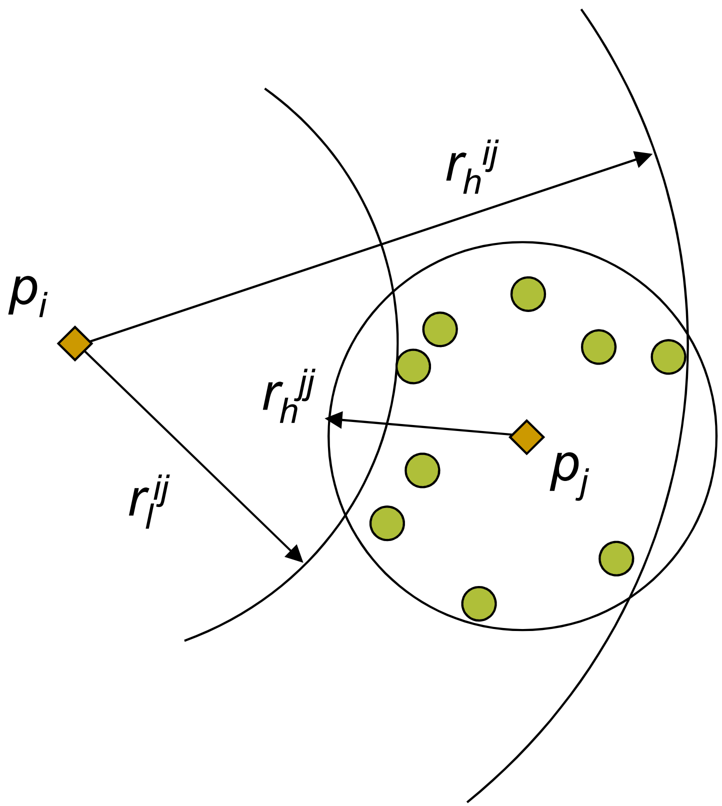
Geometric Near-neighbor Access Tree (GNAT)

- *m*-ary tree based on Voronoi-like partitioning
  - *m* can vary with the level in the tree.
- A set of pivots $P=\{p_1,\ldots,p_m\}$ is selected from X
  - Split X into m subsets S_i
  - $\forall o \in X\text{-}P$: $o \in S_i$ if $d(p_i,o) \leq d(p_j,o)$ for all $j=1..m$
  - This process is repeated recursively.



GNAT (cont.)

- Pre-computed distances are also stored.
- An $m \times m$ table of distance ranges is in each internal node.
  - Minimum and maximum of distances between each pivot p_i and the objects of each subset S_j are stored.



GNAT (cont.)

- The $m \times m$ table of distance ranges

	p_1	p_2	…	p_{m-1}	p_m
S_1	[0.0, 2.1]	[3.0, 3.8]	…	[4.2, 7.0]	[2.1, 4.0]
S_2	[2.3, 3.7]	[0.0, 1.5]		[2.8, 4.2]	[6.8, 8.3]
⋮	⋮	⋮	⋱	⋮	⋮
S_{m-1}	[5.2, 6.0]	[6.9, 7.8]	…	[0.0, 0.9]	[8.0, 8.7]
S_m	[1.0, 5.1]	[2.5, 6.4]		[5.9, 8.9]	[0.0, 4.2]

- Each range $[r_l^{ij}, r_h^{ij}]$ is defined as:
  - Notice that $r_l^{ii}=0$.

$$r_l^{ij} = \min_{o \in S_j \cup \{p_j\}} d(p_i, o)$$

$$r_h^{ij} = \max_{o \in S_j \cup \{p_j\}} d(p_i, o)$$

GNAT: Choosing Pivots

- For good clustering, pivots cannot be chosen randomly.
- From a sample *3m* objects, select *m* pivots:
  - Three is an empirically derived constant.
  - The first pivot at random.
  - The second pivot as the furthest object.
  - The third pivot as the furthest object from previous two.
    - The minimum of the two distances is maximized.
  - …
  - Until we have *m* pivots.

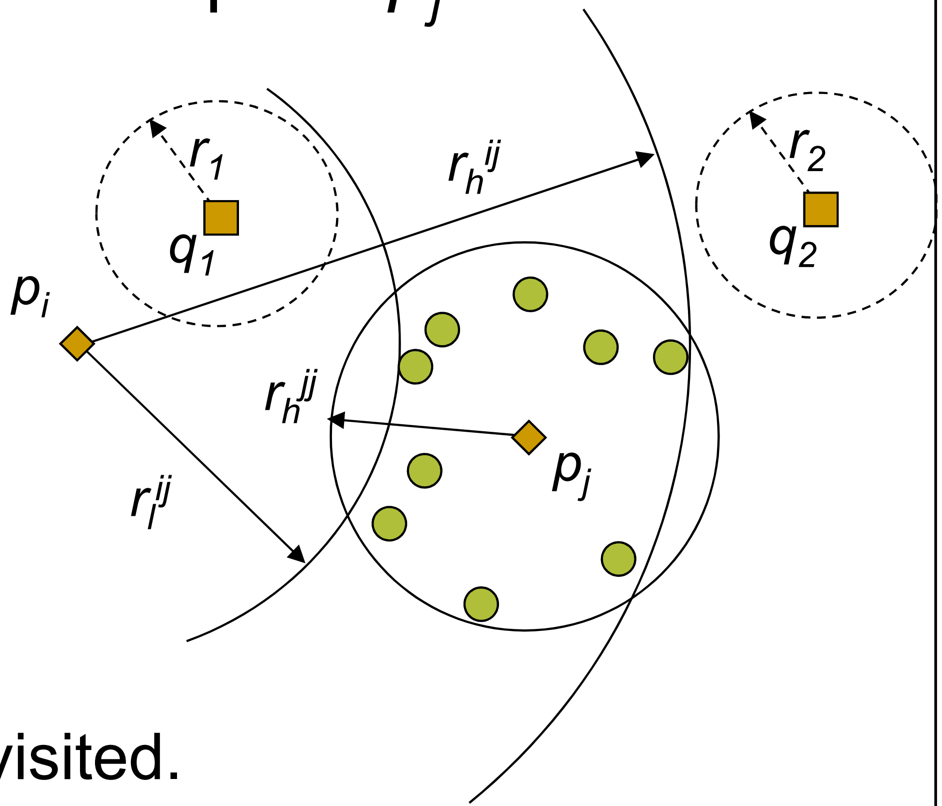
GNAT: Range Search

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

- Start in the root node and traverse the tree (depth-first).
- In internal nodes, employ the distance ranges to prune some branches.
- In leaf nodes, all objects are directly compared to *q*.
  - If $d(q,o) \leq r$, report *o* to the output.

GNAT: Range Search (cont.)

- In an internal node with pivots $p_1, p_2, \ldots, p_m$:
  - Pick one pivot p_i at random.
- Gradually pick next non-examined pivot p_j:
  - If $d(q,p_i)-r > r_h^{ij}$ or $d(q,p_i)+r < r_l^{ij}$, discard p_j and its sub-tree.
- Remaining pivots p_j are compared with q
  - If $d(q,p_j)-r > r_h^{jj}$, discard p_j and its sub-tree.
  - If $d(q,p_j) \leq r$, output p_j
  - The corresponding sub-tree is visited.

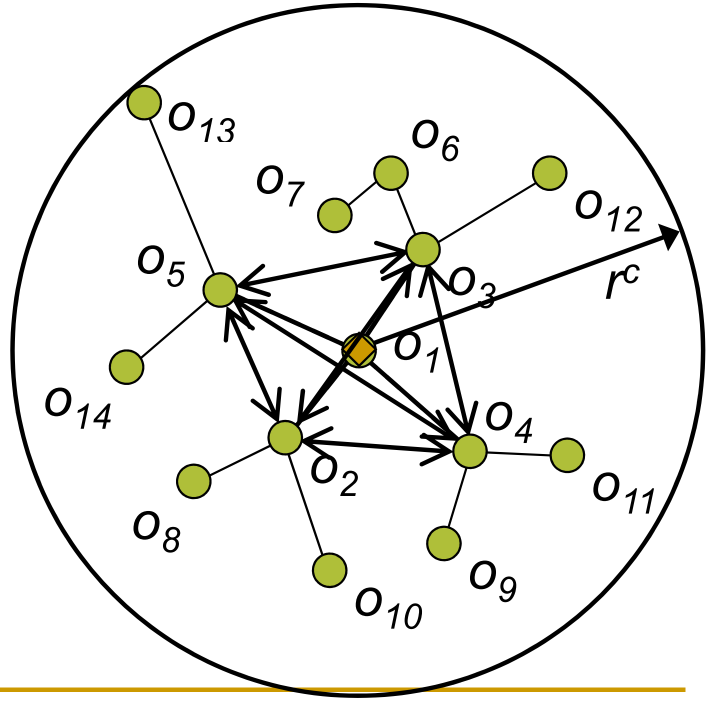


Spatial Approximation Tree (SAT)

- A tree based on Voronoi-like partitioning
  - But stores relations between partitions, i.e., an edge is between 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.
- The root *p* is a randomly selected object from *X.*
  - A set $N(p)$ of *p's* neighbors is defined
  - Every object $o \in X\text{-}N(p)\text{-}\{p\}$ is organized under the closest neighbor in $N(p)$.
  - Covering radius is defined for every internal node (object).

SAT: Example

- Intuition of $N(p)$
  - Each object of $N(p)$ is closer to p than to any other object in $N(p)$.
  - All objects in X-$N(p)$-$\{p\}$ are closer to an object in $N(p)$ than to p.
- The root is o_1
  - $N(o_1)=\{o_2,o_3,o_4,o_5\}$
  - o_7 cannot be included since it is closer to o_3 than to o_1.
  - Covering radius of o_1 conceals all objects.



SAT: Building $N(p)$

- Construction of minimal $N(p)$ is NP-complete.
- Heuristics for creating $N(p)$:
  - The pivot p, $S=X-\{p\}$, $N(p)=\{\}$.
  - Sort objects in S with respect to their distances from p.
  - Start adding objects to $N(p)$.
    - The new object o_N is added if it is not closer to any object already in $N(p)$.

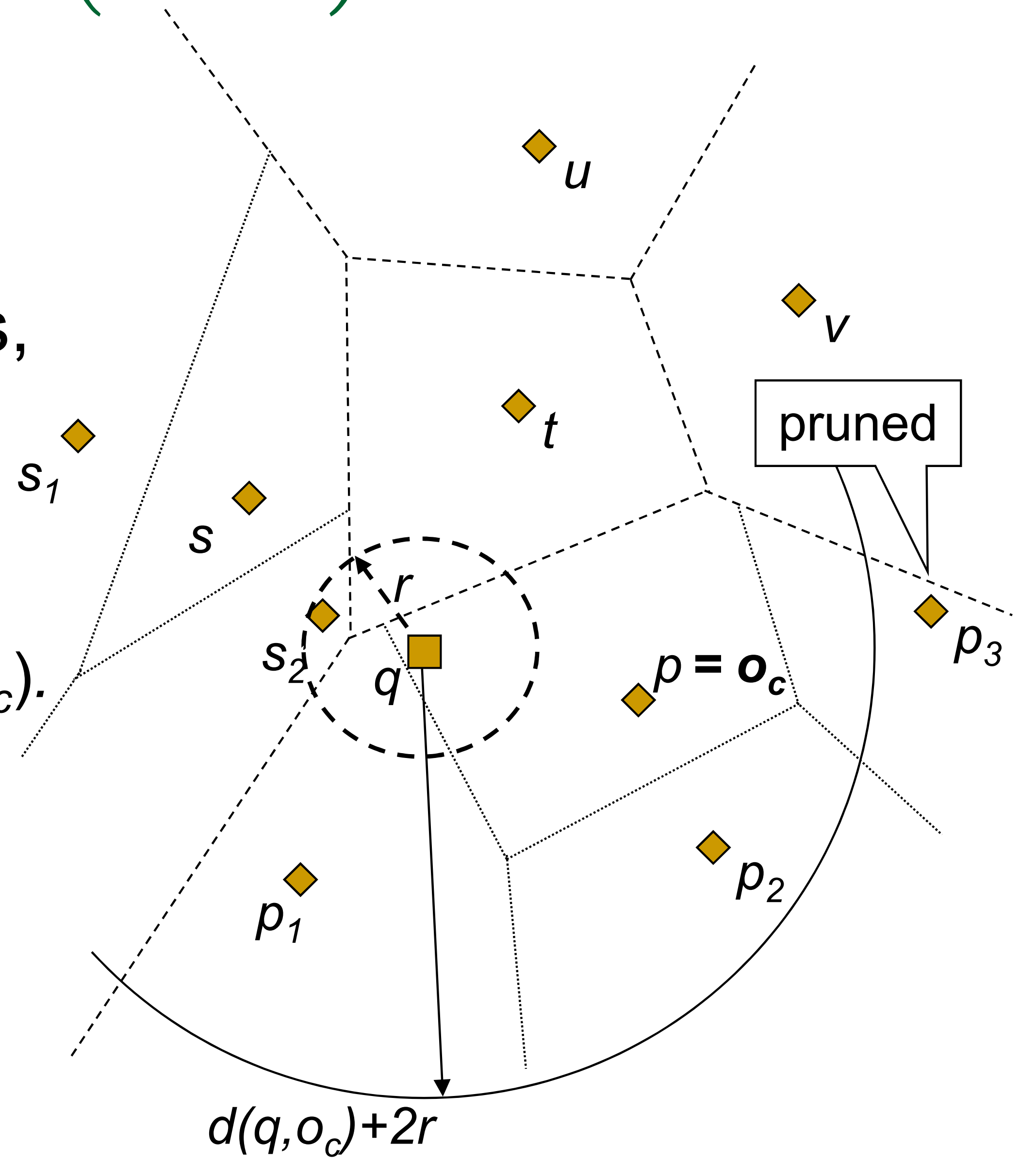
SAT: Range Search

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

- Start in the root node and traverse the tree.
- In internal nodes, employ the distance ranges to prune some branches.
- In leaf nodes, all objects are directly compared to *q*.
  - If $d(q,o) \leq r$ report *o* to the output.

SAT: Range Search (cont.)

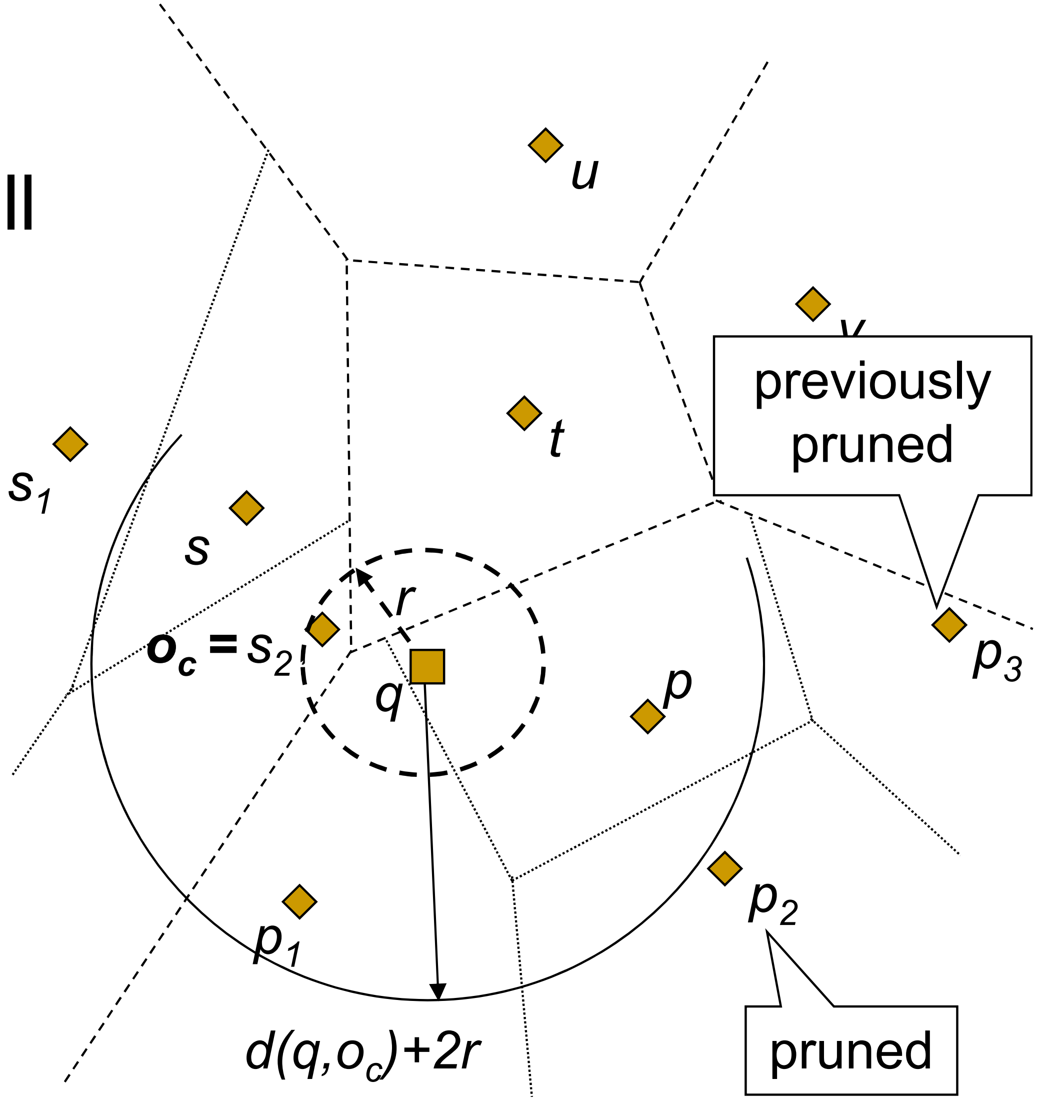
- In an internal node with the pivot p and $N(p)$:
- To prune some branches, locate the closest object $o_c \in N(p) \cup \{p\}$ to q.
  - Discard sub-trees $o_d \in N(p)$ such that $d(q,o_d) > 2r + d(q,o_c)$.
  - The pruning effect is maximized if $d(q,o_c)$ is minimal.



SAT: Range Search (cont.)

- If we pick s_2 as the closest object, pruning will be improved.
  - The sub-tree p_2 will be discarded.
- Select the closest object among more “neighbors”:
  - Use *p*’s ancestor and its neighbors.
  - $o_c \in \bigcup_{o \in A(p)} N(o) \cup \{o\}$

$$A(p) = \{t, p, s, u, v\}$$



SAT: Range Search (cont.)

- Finally, apply covering radii of remaining objects
  - Discard o_d such that $d(q,o_d)>r_d^c+r$.

M-tree

- inherently **dynamic** structure
- **disk-oriented** (fixed-size nodes)
- built in a **bottom-up** fashion

- each node constrained by a sphere-like (ball) region
- *leaf node*: data objects + their distances from a *pivot* kept in the parent node
- *internal node*: pivot + radius covering the subtree, distance from the pivot the *parent pivot*
- *filtering*: covering radii + pre-computed distances

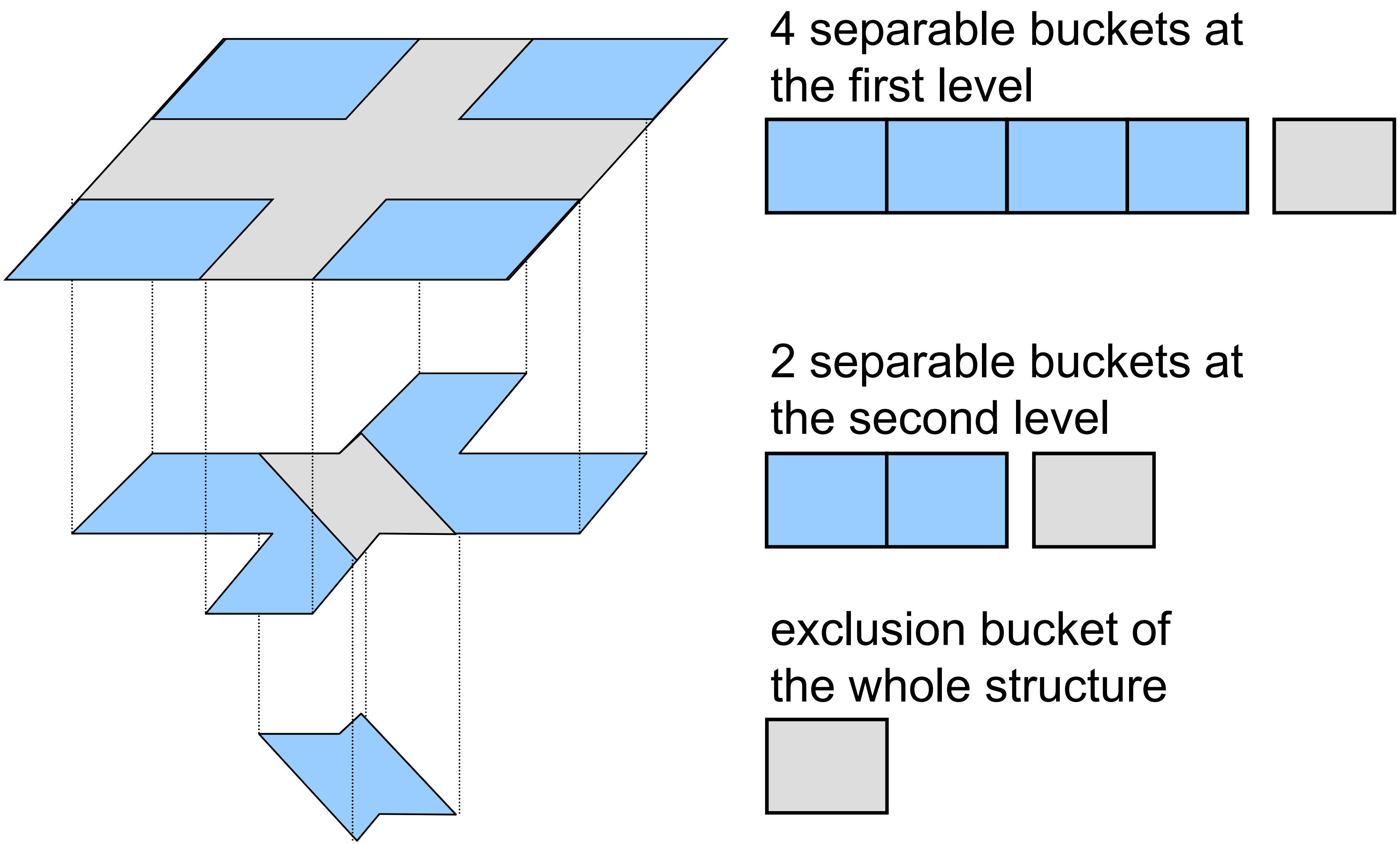
M-tree: Extensions

- bulk-loading algorithm
  - considers the trade-off: dynamic properties vs. performance
  - M-tree building algorithm for a dataset *given in advance*
  - results in more efficient M-tree
- Slim-tree
  - variant of M-tree (dynamic)
  - reduces the *fat-factor* of the tree
  - tree with smaller overlaps between particular tree regions
- many variants and extensions – see Chapter 3

Similarity Hashing

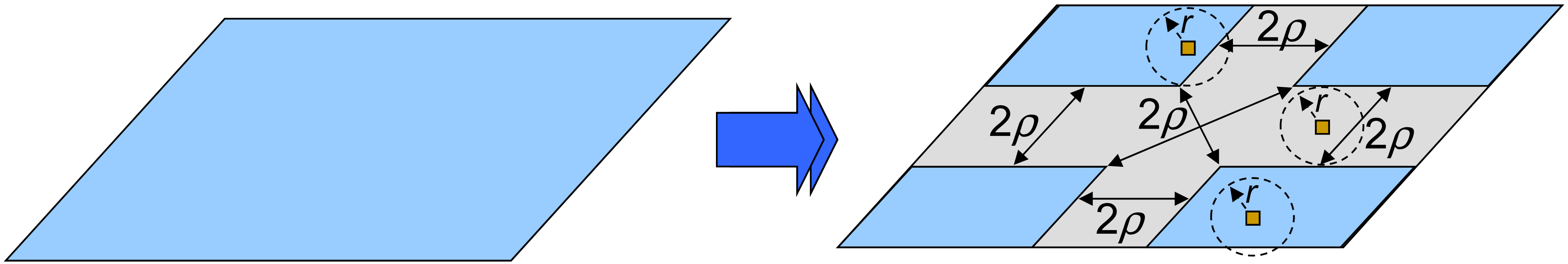
- Multilevel structure
- One hash function (ρ-split function) per level
  - Producing several buckets.
- The first level splits the whole data set.
- Next level partitions the exclusion zone of the previous level.
- The exclusion zone of the last level forms the exclusion bucket of the whole structure.

Similarity Hashing: Structure



Similarity Hashing: ρ-Split Function

- Produces several separable buckets.
  - Queries with radius up to ρ accesses one bucket at most.
  - If the exclusion zone is touched, next level must be sought.



Similarity Hashing: Features

- Bounded search costs for queries with radius ≤ ρ.
  - One bucket per level at maximum
- Buckets of static files can be arranged in a way that I/O costs never exceed the sequential scan.
- Direct insertion of objects.
  - Specific bucket is addressed directly by computing hash functions.
- D-index is based on similarity hashing.
  - Uses excluded middle partitioning as the hash function.

Survey of Existing Approaches

1. ball partitioning methods
2. generalized hyper-plane partitioning approaches
3. exploiting pre-computed distances
4. hybrid indexing approaches
5. **approximated techniques**

Approximate Similarity Search

- Space transformation techniques
  - Introduced very briefly
- Reducing the subset of data to be examined
  - Most techniques originally proposed for vector spaces
    - Some can also be used in metric spaces
  - Some are specific for metric spaces

Exploiting Space Transformations

- Space transformation techniques transform the original data space into another suitable space.
  - As an example consider dimensionality reduction.

- Space transformation techniques are typically distance preserving and satisfy the lower-bounding property:
  - Distances measured in the transformed space are smaller than those computed in the original space.

Exploiting Space Transformations (cont.)

- Exact similarity search algorithms:
  - Search in the transformed space
  - Filter out non-qualifying objects by re-measuring distances of retrieved objects in the original space.
- Approximate similarity search algorithms
  - Search in the transformed space
  - Do not perform the filtering step
    - False hits may occur

BBD Trees

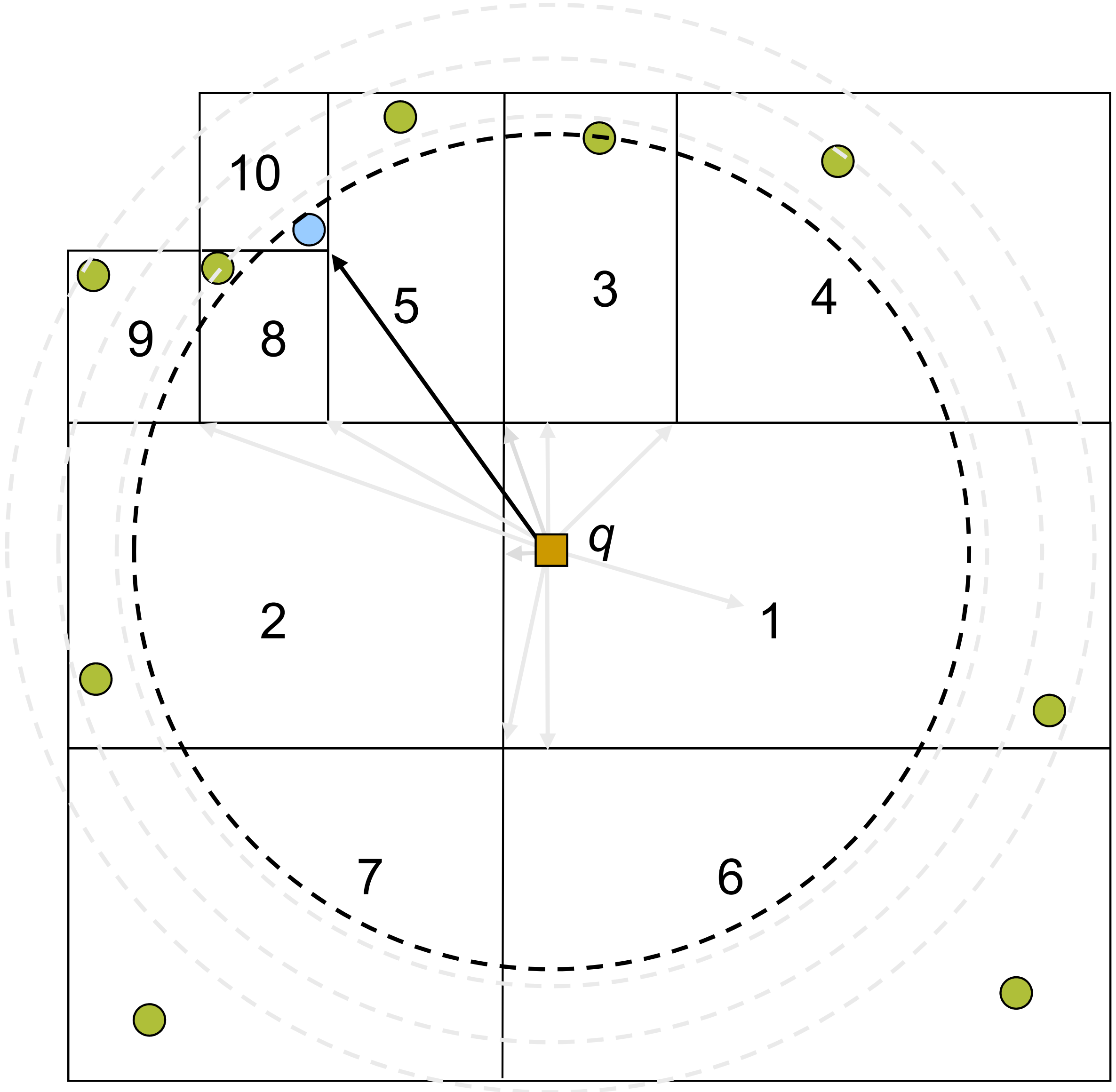
- A Balanced Box-Decomposition (BBD) tree hierarchically divides the vector space with *d*-dimensional non-overlapping boxes.
  - Leaf nodes of the tree contain a single object.
  - BBD trees are intended as a main memory data structure.

BBD Trees (cont.)

- Exact *k-NN*(*q*) search is obtained as follows
  - Find the leaf containing the query object
  - Enumerate leaves in the increasing order of distance from *q* and maintain the *k* closest objects.
  - Stop when the distance of next leaf is greater than $d(q,o_k)$.
- Approximate *k-NN*(*q*):
  - Stop when the distance of next leaf is greater than $d(q,o_k)/(1+\varepsilon)$.
- Distances from *q* to retrieved objects are at most $1+\varepsilon$ times larger than that of the *k*-th actual nearest neighbor of *q*.

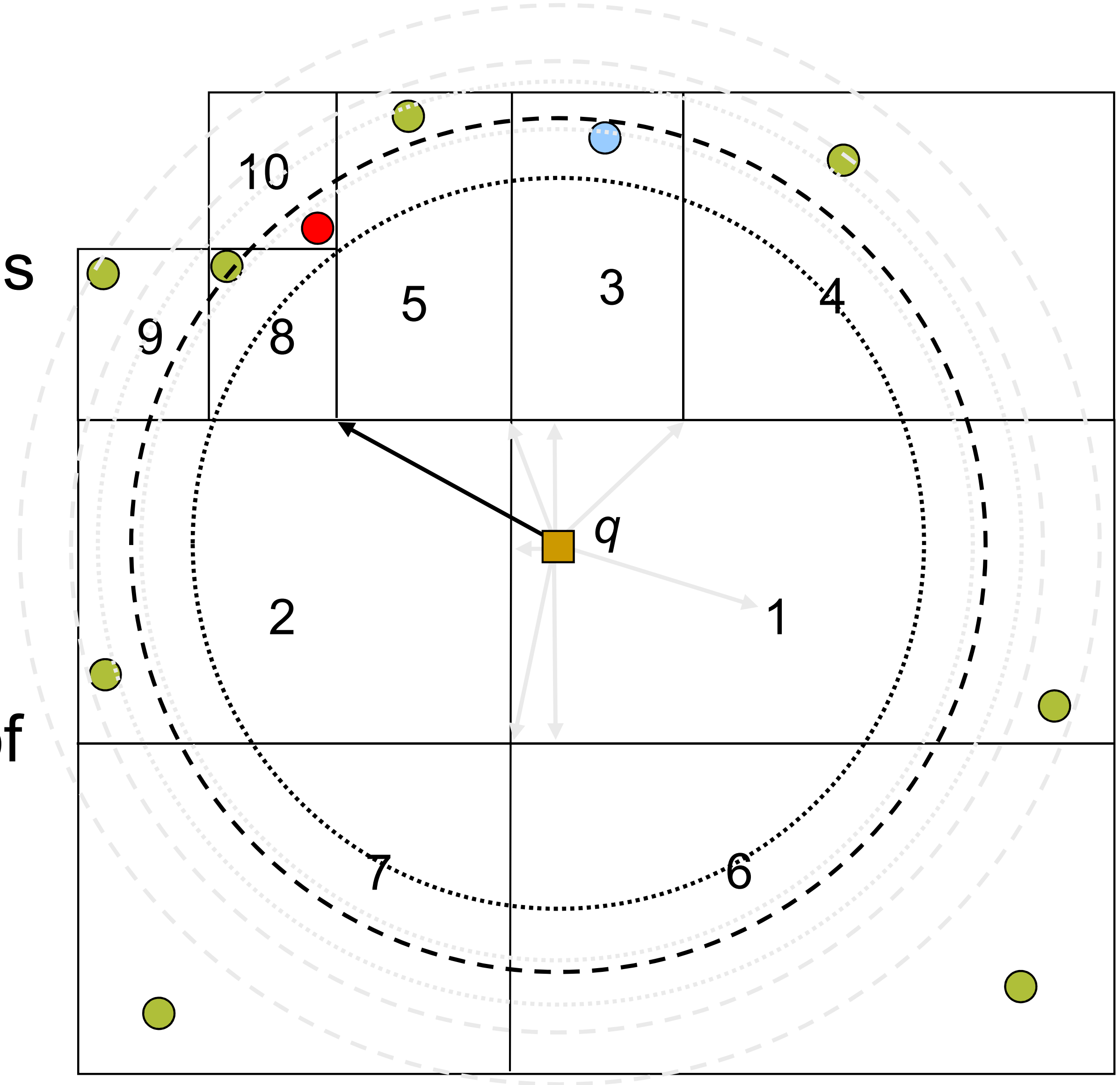
BBD Trees: Exact *1-NN* Search

- Given *1-NN(q):*



BBD Trees: Approximate *1-NN* Search

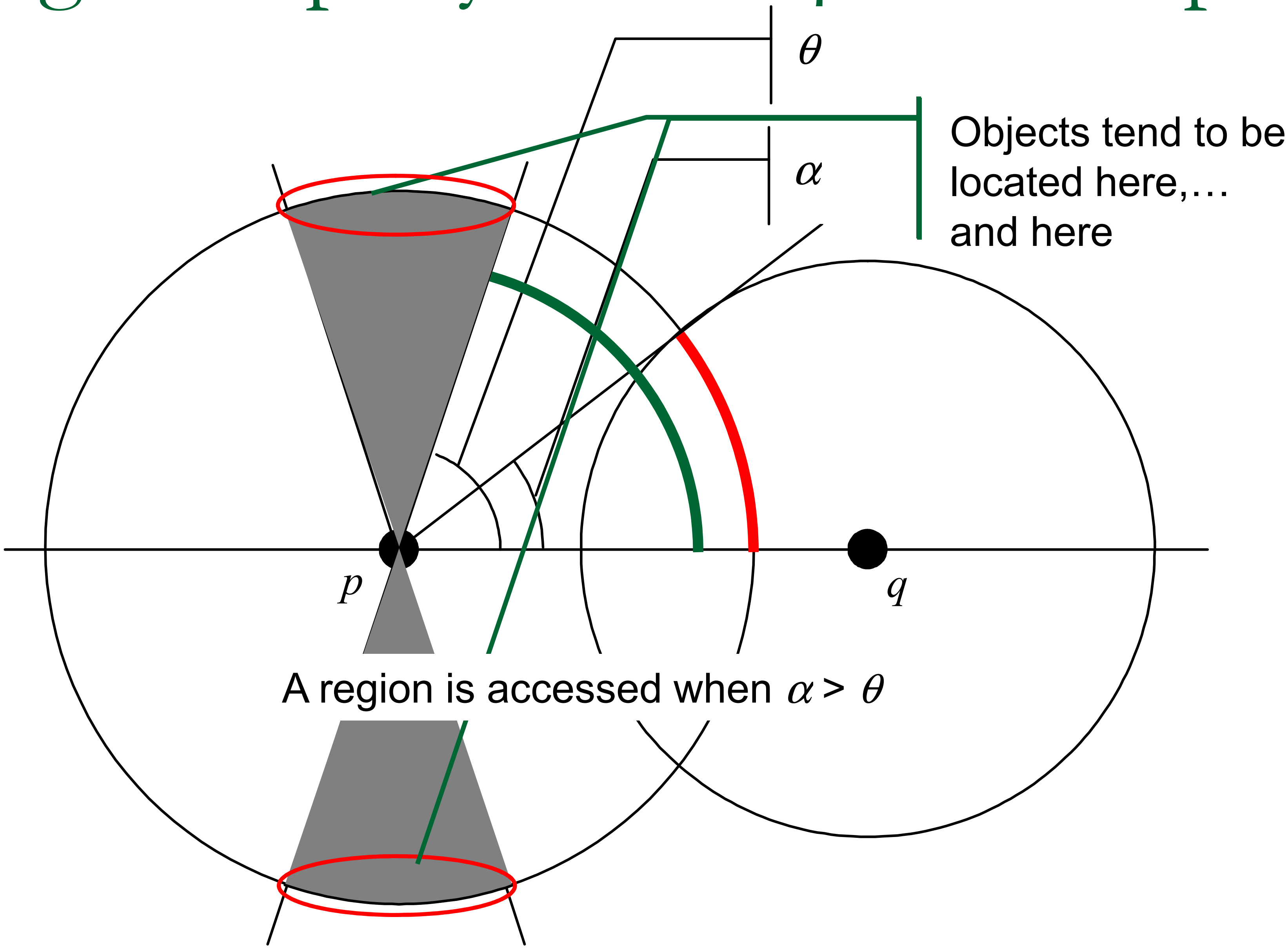
- Given *1-NN(q):*
  - Radius $d(q,o_{NN})/(1+e)$ is used instead!
- Regions 9 and 10 are not accessed:
  - They do not intersect the dashed circle of radius $d(q,o_{NN})/(1+e)$.
- The exact *NN* is missed!



Angle Property Technique

- Observed (non-intuitive) properties in high dimensional vector spaces:
  - Objects tend to have the same distance.
    - Therefore they tend to be distributed on the surface of ball regions.
  - Parent and child regions have very close radii.
    - All regions intersect one each other.
  - The angle formed by a query point, the centre of a ball region, and any data object is close to 90 degrees.
    - The higher the dimensionality, the closer to 90 degrees.
- These properties can be exploited for approximate similarity search.

Angle Property Technique: Example



Clustering for Indexing (Clindex)

- Performs approximate similarity search in vector spaces exploiting clustering techniques.
- The dataset is partitioned into clusters of similar objects:
  - Each cluster is represented by a separate file sequentially stored on the disk.

Clindex: Approximate Search

- Approximate similarity search:
  - Seeks for the cluster containing (or the cluster closest to) the query object.
  - Sorts the objects in the cluster according to the distance to the query.
- The search is approximate since qualifying objects can belong to other (non-accessed) clusters.
- More clusters can be accessed to improve precision.

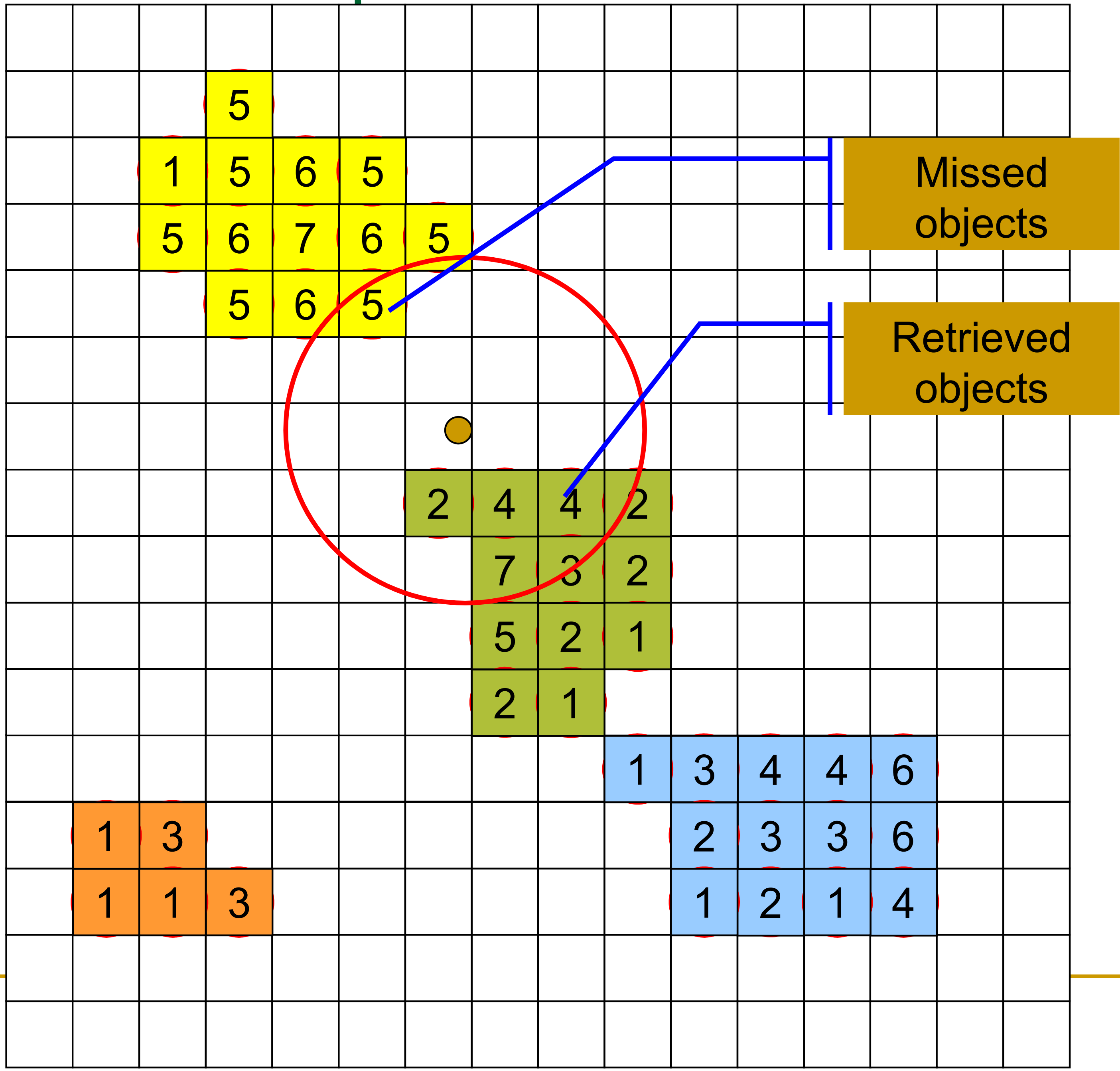
Clindex: Clustering

- Clustering:
  - Each dimension of the *d*-dimensional vector space is divided into 2^n segments: the result is $(2^n)^d$ cells in the data space.
  - Each cell is associated with the number of objects it contains.

Clindex: Clustering (cont.)

- Clustering starts accessing cells in the decreasing order of number of contained objects:
  - If a cell is adjacent to a cluster it is attached to the cluster.
  - If a cell is not adjacent to any cluster it is used as the seed for a new cluster.
  - If a cell is adjacent to more than one cluster, a heuristics is used to decide:
    - if the clusters should be merged or
    - which cluster the cell belongs to.

Clindex: Example



Vector Quantization index (VQ-Index)

- This approach is also based on clustering techniques to perform approximate similarity search.
- Specifically:
  - The dataset is grouped into (non-necessarily disjoint) subsets.
  - Lossy compression techniques are used to reduce the size of subsets.
  - A similarity query is processed by choosing a subset where to search.
  - The chosen compressed dataset is searched after decompressing it.

VQ-Index: Subset Generation

- Subset generation:
  - Query objects submitted by users are maintained in a history file.
  - Queries in the history file are grouped into *m* clusters by using *k-means* algorithm.
  - In correspondence of each cluster C_i a subset S_i of the dataset is generated as follows

$$S_i = \bigcup_{q \in C_i} kNN(q)$$

  - An object may belong to several subsets.

VQ-Index: Subset Generation (cont.)

- The overlap of subsets versus performance can be tuned by the choice of *m* and *k*
  - Large *k* implies more objects in a subset, so more objects are recalled.
  - Large values of *m* implies more subsets, so less objects to be accessed.

VQ-Index: Compression

- Subset compression with vector quantisation:
  - An encoder *Enc* function is used to associate every vector with an integer value taken from a finite set {1,…,*n*}.
  - A decoder *Dec* function is used to associate every number from the set {1,…,*n*} with a representative vector.
  - By using *Enc* and *Dec,* every vector is represented by a representative vector
    - Several vectors might be represented by the same representative.
  - *Enc* is used to compress the content of S_i by applying it to every object in it:

$$S_i^{enc} = \{Enc_i(x) \mid x \in S_i\}$$

VQ-Index: Approximate Search

- Approximate search:
  - Given a query *q*:
  - The cluster C_i closest to the query is first located.
  - An approximation of S_i is reconstructed, by applying the decoder function Dec_i .
  - The approximation of S_i is searched for qualifying objects.
  - Approximation occurs at two stages:
    - Qualifying objects may be included in other subsets, in addition to S_i .
    - The reconstructed approximation of S_i may contain vectors which differ from the original ones.

Buoy Indexing

- Dataset is partitioned in disjoint clusters.
- A cluster is represented by a representative element – the *buoy.*
- Clusters are bounded by a ball region having the buoy as center and the distance of the buoy to the farthest element of the cluster as the radius.
- This approach can be used in pure metric spaces.

Buoy Indexing: Similarity Search

- Given an exact *k-NN* query, clusters are accessed in the increasing distance to their buoys, until current result-set cannot be improved.
  - That is, until $d(q,o_k) + r_i < d(q,p_i)$
    - p_i is the buoy, r_i is the radius
- An approximate *k-NN* query can be processed by stopping when
  - either previous exact condition is true, or
  - a specified ratio *f* of clusters has been accessed.

Hierarchical Decomposition of Metric Spaces

- In addition to previous ones, there are other methods that were appositively designed to
  - Work on generic metric spaces
  - Organize large collections of data
- They exploit the hierarchical decomposition of metric spaces.

Hierarchical Decomposition of Metric Spaces (cont.)

- These will be discussed in details later on:
  - Relative error approximation
    - Relative error on distances of the approximate result is bounded.
  - Good fraction approximation
    - Retrieves *k* objects from a specified fraction of the objects closest to the query.

Hierarchical Decomposition of Metric Spaces (cont.)

- These will be discussed in details later on:
  - Small chance improvement approximation
    - Stops when chances of improving current result are low.
  - Proximity based approximation
    - Discards regions with small probability of containing qualifying objects.
  - PAC (Probably Approximately Correct) nearest neighbor search
    - Relative error on distances is bounded with a probability specified.