INFS4205/7205 Advanced Techniques for High Dimensional Data

Managing High-Dimensional Data

Semester 1, 2021

University of Queensland

+ Advanced Techniques for High Dimensional Data

- Course Introduction
- Introduction to Spatial Databases
- Spatial Data Organization
- Spatial Query Processing
- Managing Spatiotemporal Data
- Managing High-Dimensional Data
- Introduction to Multimedia Database
- Route Planning in Road Network
- When AI Meets High-Dimensional Data
- Trends and Course Review

+ Outline

- Motivation
  - Examples
  - Why
  - What to expect
- Technique
  - X-Tree
  - Pyramid
  - VA-File
  - iDistance
  - Other techniques
- Summary

+ Outline

- Motivation
  - Examples
  - Why
  - What to expect
- Technique
  - X-Tree
  - Pyramid
  - VA-File
  - iDistance
  - Other techniques
- Summary

+ High-Dimensional Feature Space

- Picture elements in digital images
- Represented by <red, green, blue>
  - Black: <0, 0, 0>
  - Orange: <255,165,0>

Red	Green	Blue	Pixel Count
0	0	0	7414
0	0	1	230
0	0	2	0
0	0	3	0
0	1	0	8
0	1	1	372
0	1	2	88
0	1	3	0
0	2	0	0
0	2	1	0
0	2	2	10

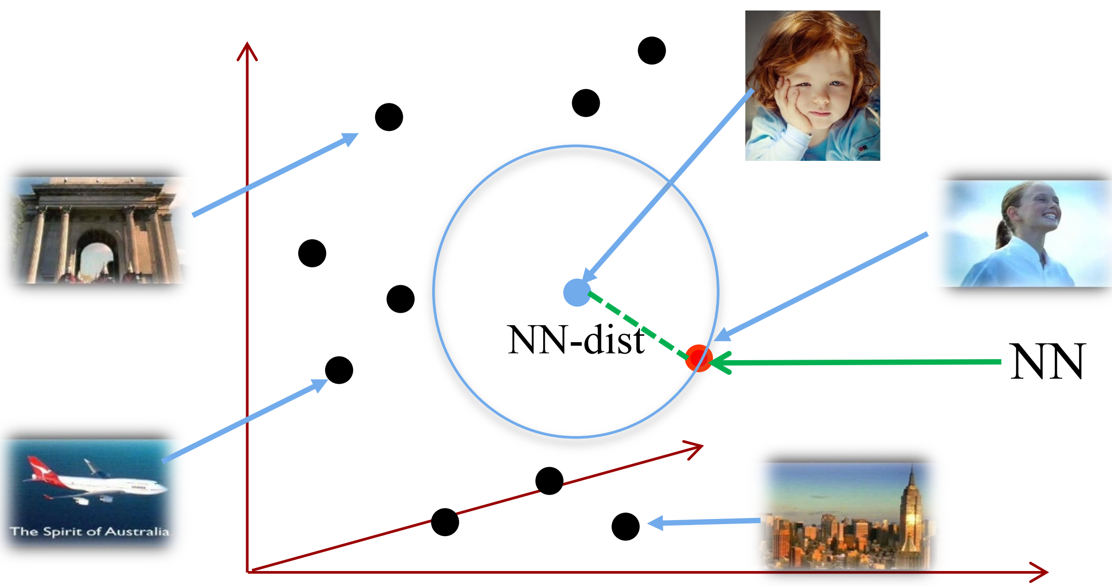
$$\text{Img} = <7414,\ 230,\ 0,\ 0,\ \ldots >$$

$$2^8 = 256\text{D color space}$$

+ High-Dimensional Feature Space

f_1 : <0.336,0.130,0.023,0.331,0.132,0.000,0.120,0.181>
f_2 : <0.331,0.123,0.028,0.338,0.008,0.011,0.132,0.181>
f_3 : <0.331,0.116,0.028,0.345,0.101,0.179,0.133,0.181>
……
f_n : <0.331,0.102,0.021,0.336,0.009,0.000,0.009,0.192>

High-dimensional data points



High-D Feature Space

+ Outline

- Motivation
  - Examples
  - Why
  - What to expect
- Technique
  - X-Tree
  - Pyramid
  - VA-File
  - iDistance
  - Other techniques
- Summary

+ CPU Cost for Indexing is High

- Indexing: to locate the data quickly
  - Query processing time should not increase linearly with the DB size
  - I/O cost: Disk page accesses
  - A balanced tree structure can reduce search time (I/O costs)
- However, for high-dimensional data
  - CPU cost: Computation of similarity/distance
  - The CPU cost for some basic operation is no longer negligible
  - As dimensionality increases, the portion of CPU cost in total response time increases
  - This is different from traditional databases, where only I/O costs are considered

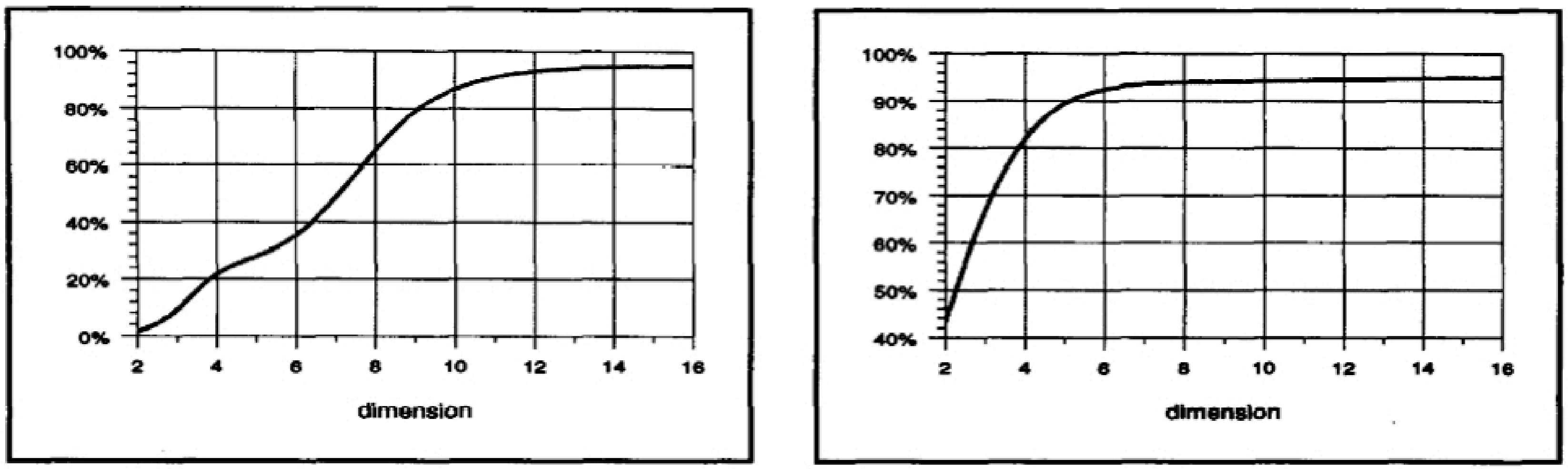
+ Indexing vs. Linear Scan

- The performance of an index degrades rapidly as dimensionality increases, and eventually underperforms linear scan!
- However, linear scan needs to search the whole data file - affected volume is 100%
  - Then query processing time will increase at least linearly with the DB size
- When the size of dataset is very large, loading the whole data into memory is unlikely
  - Even if this is possible, it's still too expensive to scan all the data

+ Overlap in R* Tree

- Aims to minimize the overlap
  - Overlap: more than one branch need to be expanded

- Dimensionality and Overlap



a. Overlap (Uniformly Distributed Data) b. Weighted Overlap (Real Data)

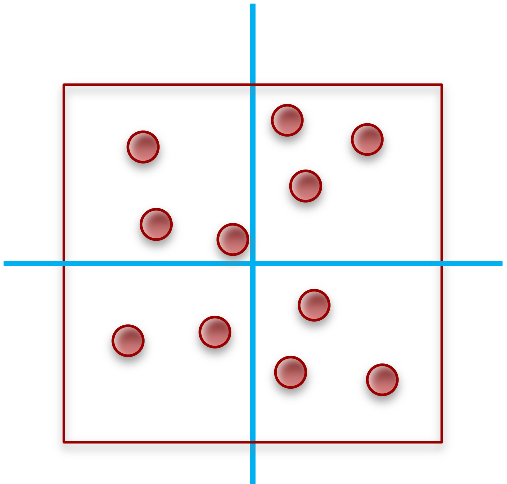
Figure 2: Overlap of R*-tree Directory Nodes depending on the Dimensionality

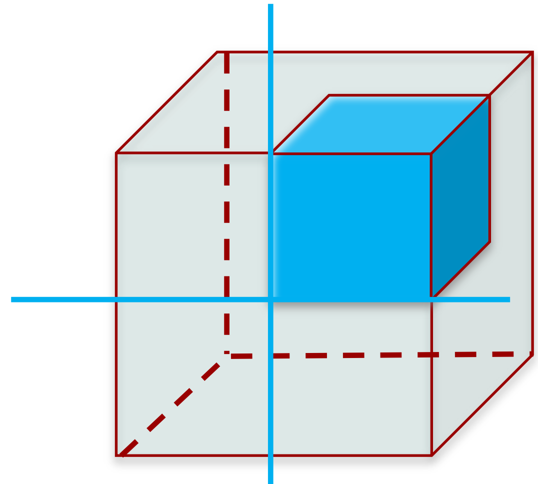
- (Overlap: % of space covered by more than one R*-tree node)
- (Weighted Overlap: % of data objects in overlapping space)

+ "Curse of Dimensionality"

- Too many partitions
- Too few points
- The nearest is not near enough

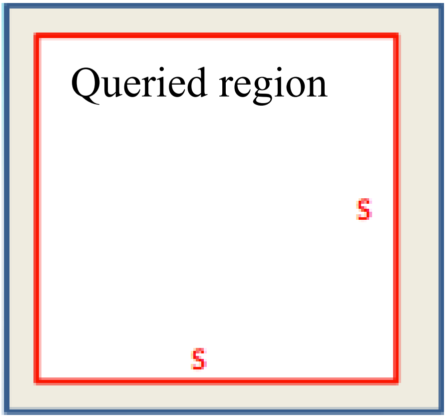
+ Too Many Partitions



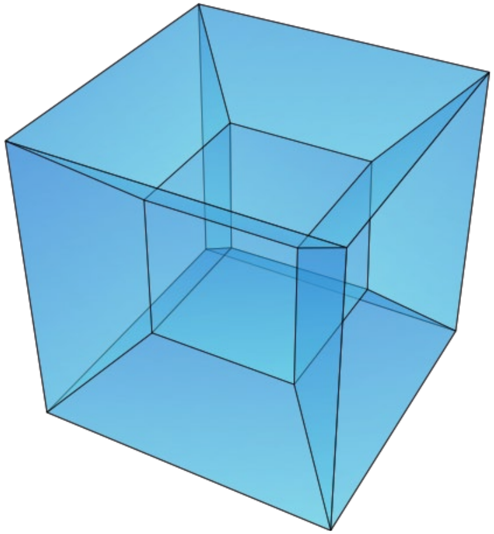


- A simple partitioning scheme splits the data space in each dimension into 2 parts
  - Thus, d-dimensions implies 2^d partitions
- For a 1,000,000 (10^6) data points database:
  - If $d \leq 10$, at most 1024 partitions for 1M points
  - If d is large, e.g., $d = 100$, there are around 2^{100} (about 10^{30}) partitions for 10^6 data points (much more partitions than points!)
- An overwhelming majority of the partitions are empty!

+ Too Few Points (1)



Data space = $[0,1]^d$



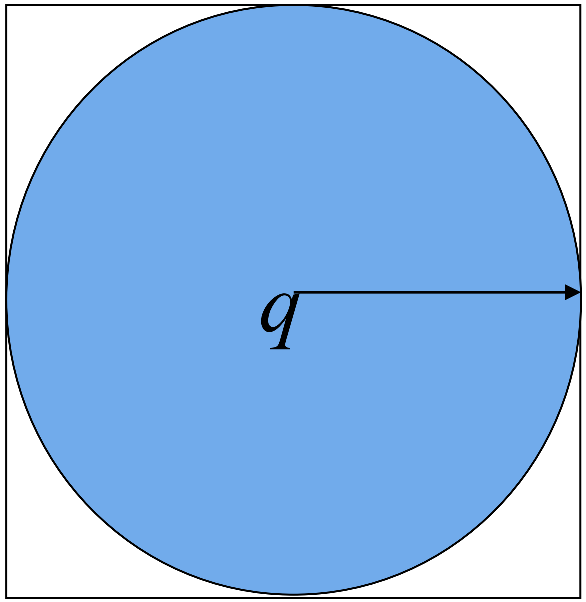
- Assume the data space is a hypercube, where each dimension is within the range of [0,1]
- The probability of a uniformly distributed point p lying within a hypercube range query with side length s is
$$Pr[p \in \text{QueriedRegion}] = s^d$$
- For uniform data, when $d = 100$, a hypercube range query with $s = 0.95$ only covers 0.59% of the data points; compared to $d = 2$, 90.25%

+ Too Few Points (2)

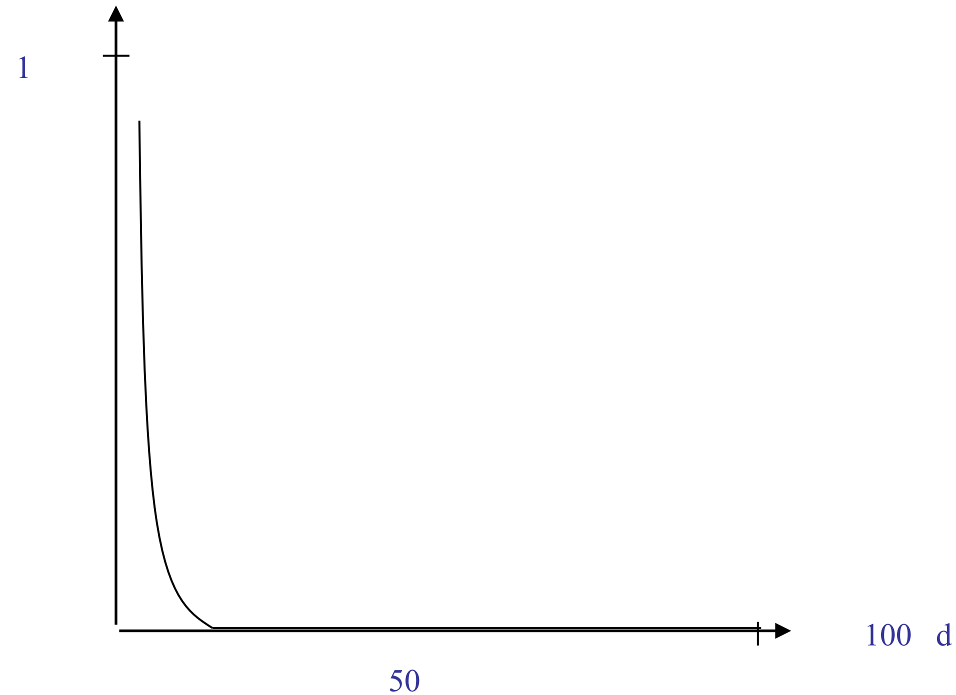
- Most of the data lies close to the boundary of the dataspace
- Suppose we have a 50-dimensional hypercube with length of 1
  - Total $\text{Volume} = 1 \times 1 \times 1 \times \cdots \times 1 = 1^{50} = 1$
  - If we regard the inner 90% of each dimension as the interior region, and the outside of it as boundary region
    - $0.00 < x_1 < 0.90,\ 0.00 < x_2 < 0.90,\ \ldots,\ 0.00 < x_{50} < 0.90$
  - If the data is uniformly distributed
    - The interior region's volume is $0.9^{50} \approx 0.005$
    - The boundary region's volume is 1-0.005 = 0.995
    - 99.5% of the points are in the boundary region

+ Too Few Points (3)

- The probability of a uniformly distributed point lying within a spherical query $Sphere(q, 0.5)$ is:



- $\boldsymbol{d = 2}$: $\pi r^2/(2r)2 = \pi/4 = 0.785$
- $\boldsymbol{d = 3}$: $(\frac{4\pi}{3} \times r^3)/(2r)^3 = \pi/6 = 0.524$
- …
- $\boldsymbol{d = 8}$: $(\frac{\pi^4}{24} \times r^8)/(2r^8) = \pi^4/(3 \times 2^{11}) = 0.016$



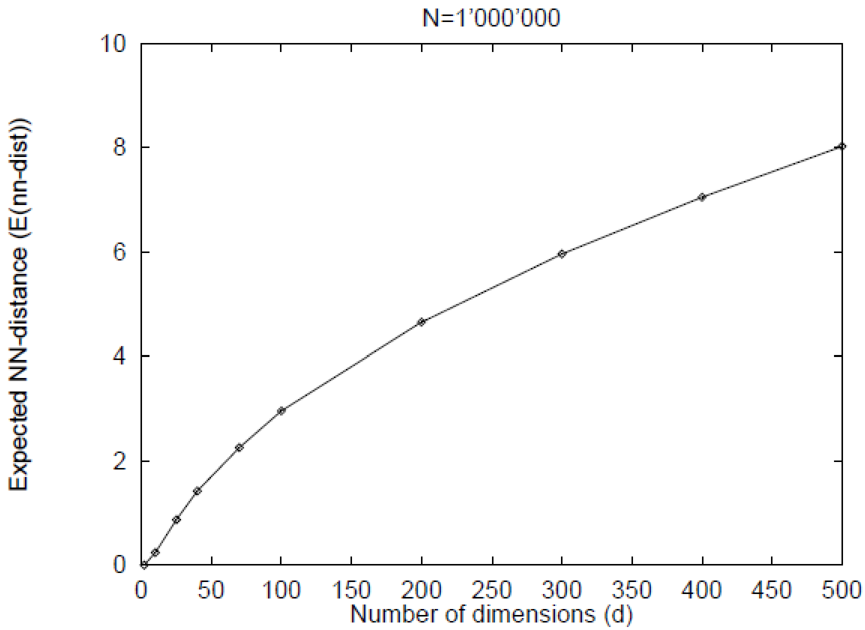
- Note that $Sphere(q, 0.5)$ is the largest spherical query that fits within the unit data space

+ The Nearest is not Near Enough

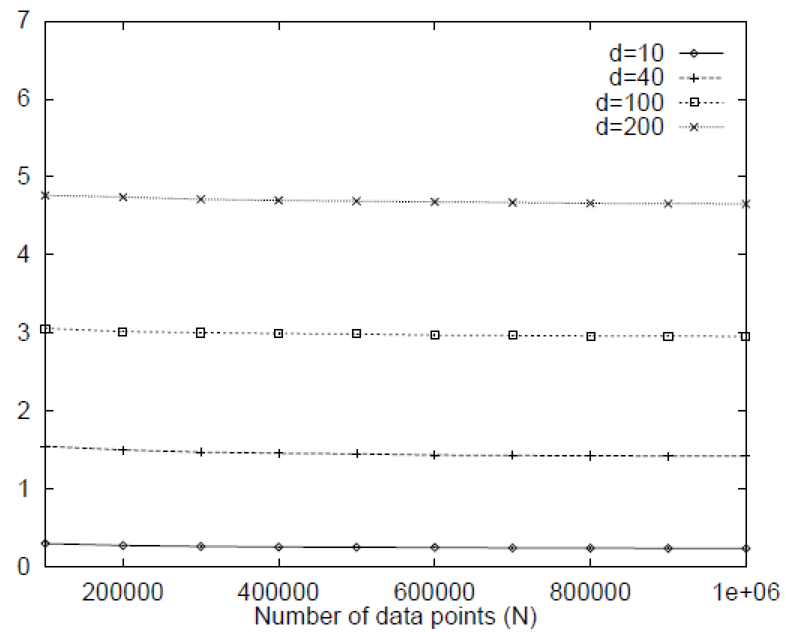
- In some sense, nearly all of the uniformly distributed high-dimensional points are **"far away"** from the centre
  - The high-dimensional data are almost entirely in the "corners" of the hypercube, with almost nothing in the "middle".
- Low distance contrast: as the dimensionality increases, the distance to the nearest neighbour **approaches** the distance to the furthest neighbour.
  - For uniformly distributed data, difference between the nearest data point and the furthest data point reduces greatly

+ The Nearest is not Near Enough (2)

- Expected NN distance vs.



dimensionality



number of data points

+ Question:

- Is the nearest neighbour meaningful in a high-D space?

- It depends on the data distribution
  - Nothing can be done for uniformly distributed data.
  - Generally, real data are not uniformly distributed, but exhibit certain clusters, trends or skewness.

+ Outline

- Motivation
  - Examples
  - Why
  - What to expect
- Technique
  - X-Tree
  - Pyramid
  - VA-File
  - iDistance
  - Other techniques
- Summary

+ Design Principles

- Simple
  - Simple in design
  - Easy to be integrated into existing DBMS
    - It’s very hard to add a new indexing structure into a DBMS
    - Better chance to be practically useful if building on top of a mature and commercially available indexing method (e.g.. B+-tree & R-tree)
- Efficient
  - Efficient in disk access (I/O cost) /CPU time
  - Must support efficient updates (insert/delete/update operations)

+ Outline

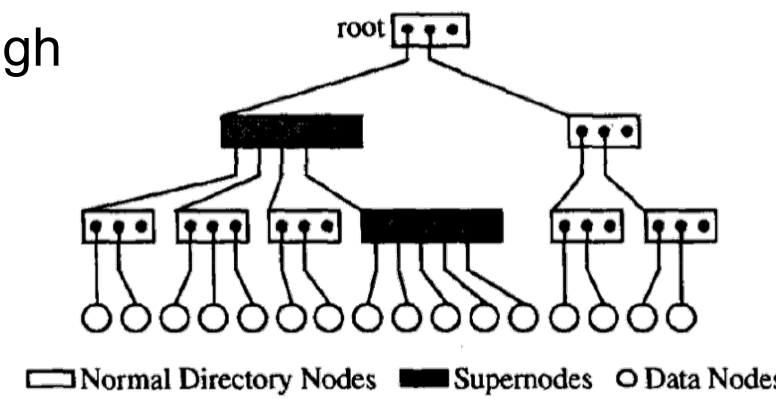
- Motivation
  - Examples
  - Why
  - What to expect
- Technique
  - X-Tree
  - Pyramid
  - VA-File
  - iDistance
  - Other techniques
- Summary

+ Outline

- Motivation
  - Examples
  - Why
  - What to expect
- Technique
  - X-Tree
  - Pyramid
  - VA-File
  - iDistance
  - Other techniques
- Summary

+ X-Tree

- Variation of R-Tree and compromise between the hierarchy and sequential access
- Avoid overlap in the directory, when we cannot find a good partition
  - In high dimensional, linear organization of the directory sometimes is more efficient
    - Due to the high overlap, most of the directory has to be searched
    - So don't split when its overlap is high
- Tree Nodes
  - Data Node
  - Normal Directory Node
  - Super Node
    - Large directory nodes of variable size
    - Arbitrary number of blocks
    - Avoid split in the directory that would result in an inefficient directory



[1] The X-tree: An Index Structure for High-Dimensional Data, VLDB, 1996

+ X-Tree

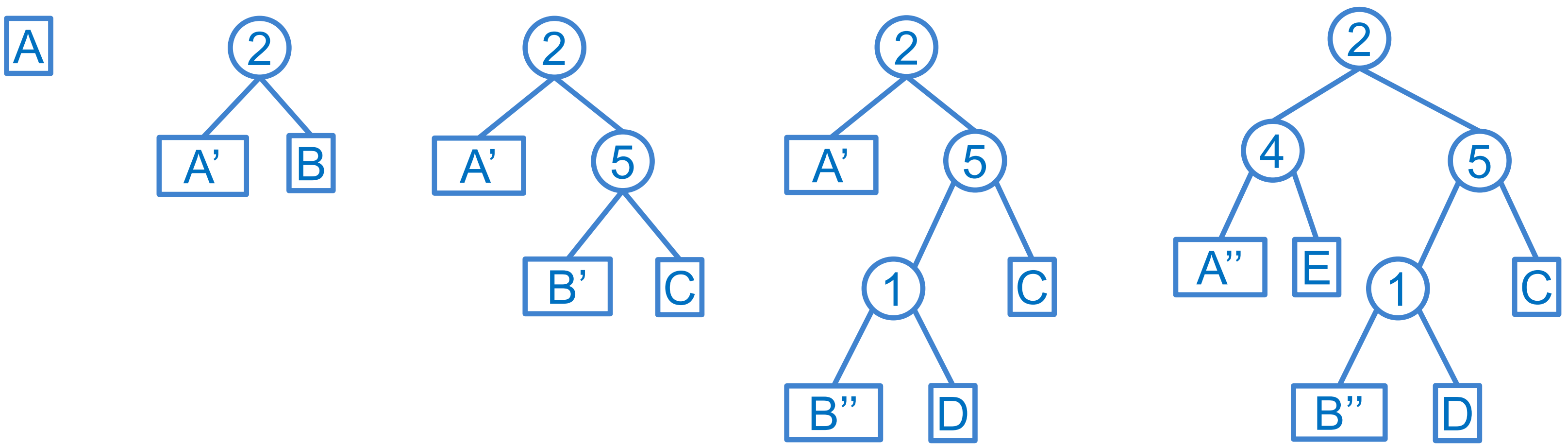
- Insertion
  - If no split occurs, update the size of MBR
  - If split is required
    1. If the overlap is low
       - Split the node with same techniques as R-tree Based techniques (or other techniques)
    2. Otherwise (the overlap is high), <u>do overlap minimal split</u>
       - If the number of MBRs in one of the partitions is below a given threshold (unbalanced)
         - Stop splitting and extend to a Supernode

+ Overlap Minimal Split*

- Split History
  - The dimension according to which an MBR has been split
  - Which new MBRs have been created by this split
- Overlap Minimal Split
  - For point data
    - It is always possible to find an overlap-free split
    - It is not possible to guarantee that the two sets are balanced
    - Determine a dimension according to which all MBRs have been split previously

+ Overlap Minimal Split for Point Data*

- Split Tree
  - Leaf Node: MBR
  - Internal Node: Nodes that have been split into new MBRs
    - Record the dimension that was used to split



    - All MBRs in any split tree have one split dimension in common: the root node

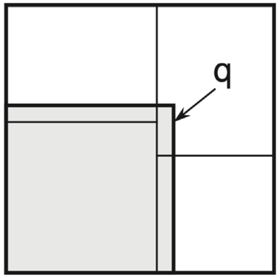
Details: [1] The X-tree: An Index Structure for High-Dimensional Data, VLDB, 1996

+ Outline

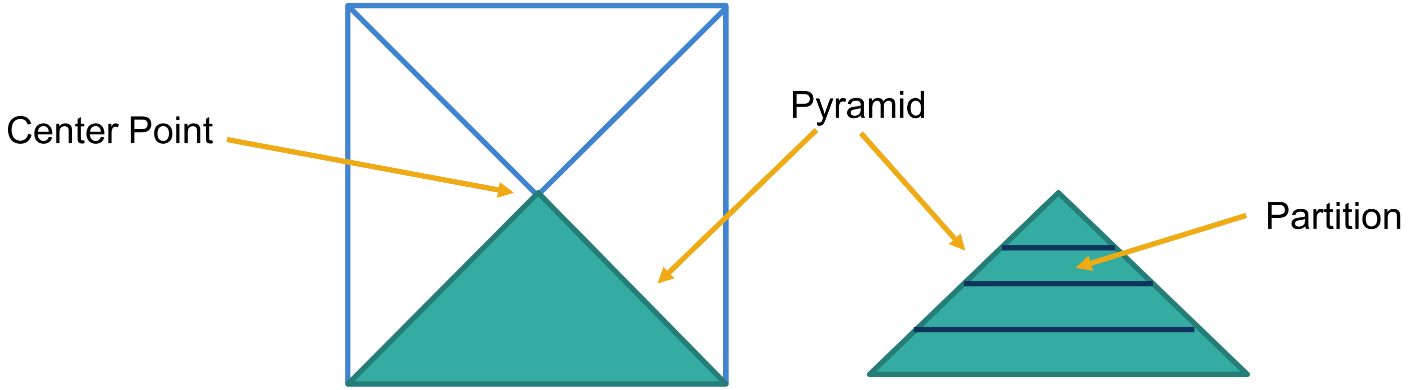
- Motivation
  - Examples
  - Why
  - What to expect
- Technique
  - X-Tree
  - Pyramid
  - VA-File
  - iDistance
  - Other techniques
- Summary

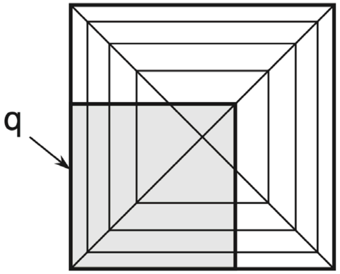
+ Pyramid Technique

- Divide the data space such that the resulting partitions are shaped like peels of an onion
  - Divide the d-D space into $2d$ pyramids having the center point (0.5, 0.5,…, 0.5) of the space as their top
  - Each pyramid is cut into slices parallel to the basis
  - Data is mapped from d-dimensional space to 1-dimensional space
  - B+-tree can then be used
  - Such that typical window query will not overlap all the nodes as the balanced split



Balanced split





Pyramid split

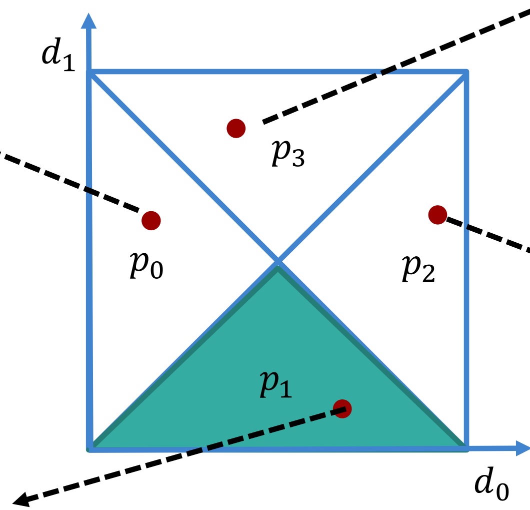
[1] The Pyramid -Technique: Towards Breaking the Curse of Dimensionality, SIGMOD, 1998

+ Pyramid Numbering

- Pyramid Numbering: a point v belongs to pyramid i
- Determine i
  - j_{max}:The dimensional j that has the maximum deviation $|0.5 - v_j|$
  - $i = \begin{cases} j_{max} \quad ,if\ v_{j_{max}} < 0.5 \\ j_{max} + d,,if\ v_{j_{max}} \geq 0.5 \end{cases}$

$$|v_1 - 0.5| < |v_0 - 0.5| \\ j_{max} = 0 \\ v_{j_{max}} = v_0 < 0.5 \\ i = j_{max} = 0$$

$$|v_1 - 0.5| > |v_0 - 0.5| \\ j_{max} = 1 \\ v_{j_{max}} = v_1 \geq 0.5 \\ i = j_{max} + d = 1 + 2 = 3$$

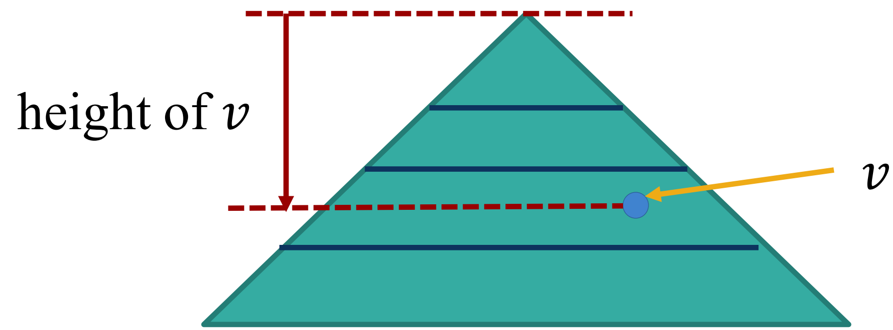


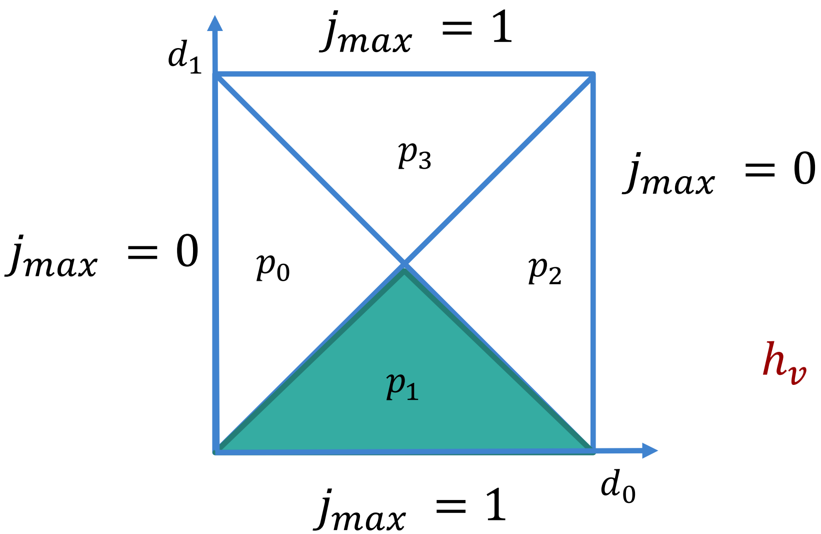
$$|v_1 - 0.5| > |v_0 - 0.5| \\ j_{max} = 1 \\ v_{j_{max}} = v_1 < 0.5 \\ i = j_{max} = 1$$

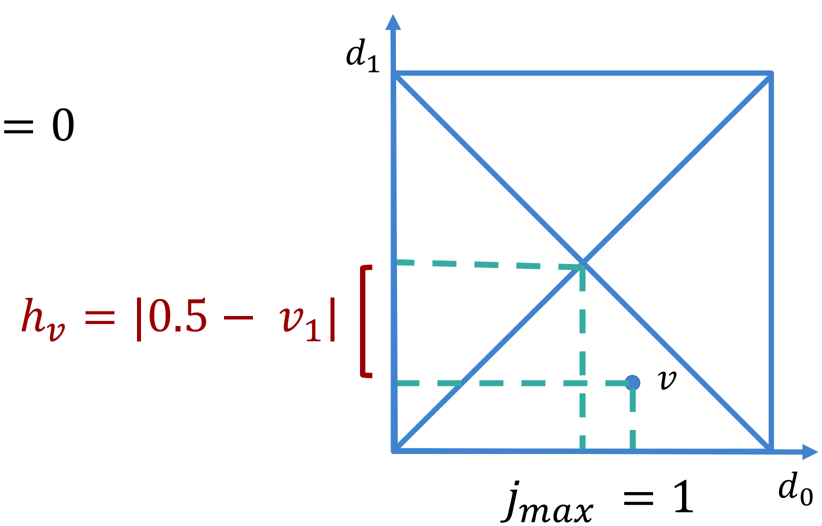
$$|v_1 - 0.5| < |v_0 - 0.5| \\ j_{max} = 0 \\ v_{j_{max}} = v_0 > 0.5 \\ i = j_{max} + d = 2$$

+ Point Height in a Pyramid

- Height of v
  - The distance from the point to the center according to dimension j_{max}
    - $h_v = |0.5 - v_{i \bmod d}|$

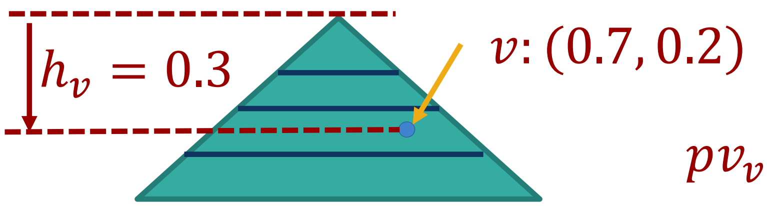




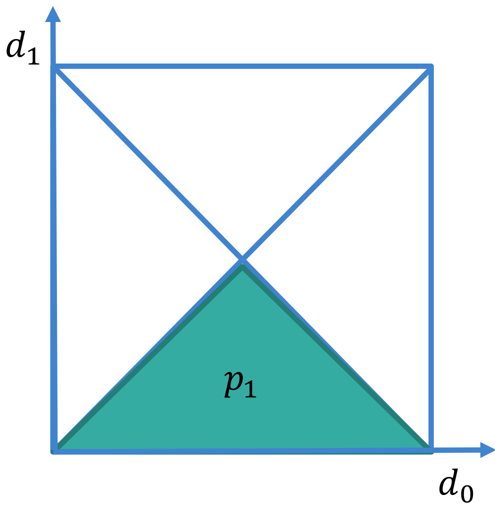


+ Pyramid Value

- Pyramid Value of d-dimensional point v
  - Transform a d-dimensional point to a value $pv_v = i + h_v$
    - i is an integer
    - h_v is in range $[0, 0.5]$
    - pv_v is in range $[i, i + 0.5]$
  - Values covered by different pyramids are disjoint



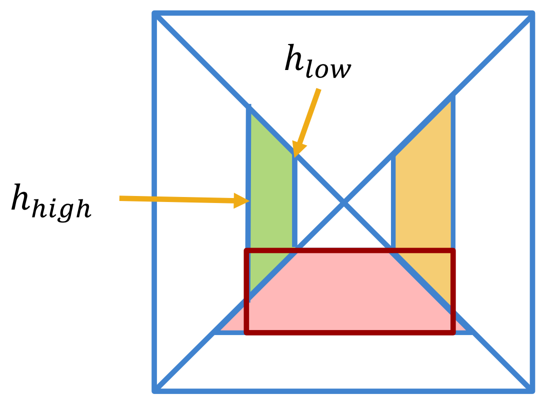
$pv_v = 1 + 0.3 = 1.3$



  - Use B+-Tree to index
    - Easy to insert, delete, and update

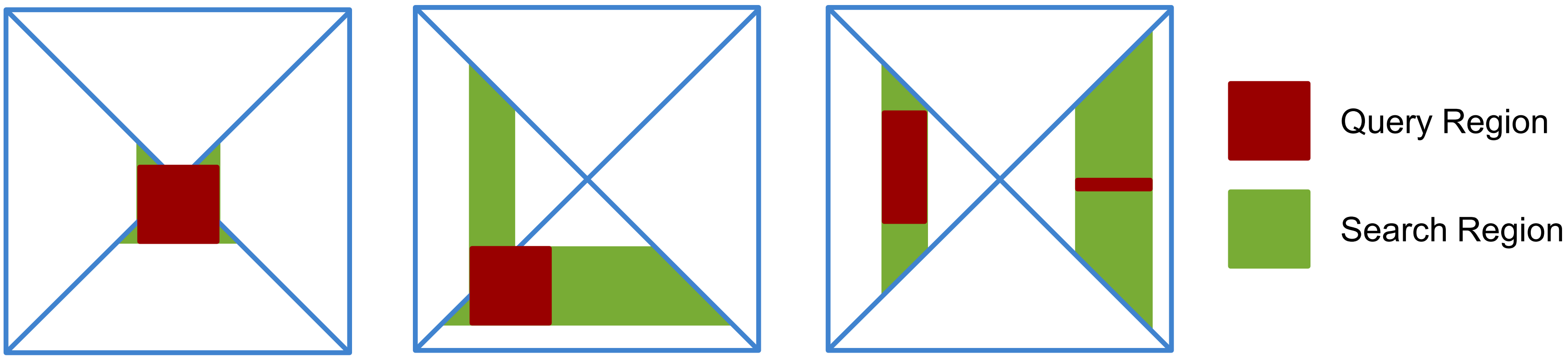
+ Pyramid Technique

- Query is complex
  - Point Query: Compute the pyramid value and query the B⁺-Tree
  - Range Query $[q_{0_{min}}, q_{0_{max}}], \ldots, [q_{d-1_{min}}, q_{d-1_{max}}]$
    1. Determine the affected pyramids
       - Transform one d-dimensional range query q into an equivalent $2d$ range queries, one for each pyramid
    2. Determine the ranges inside the pyramids
       - $[i + h_{low}, i + h_{high}]$ for each pyramid

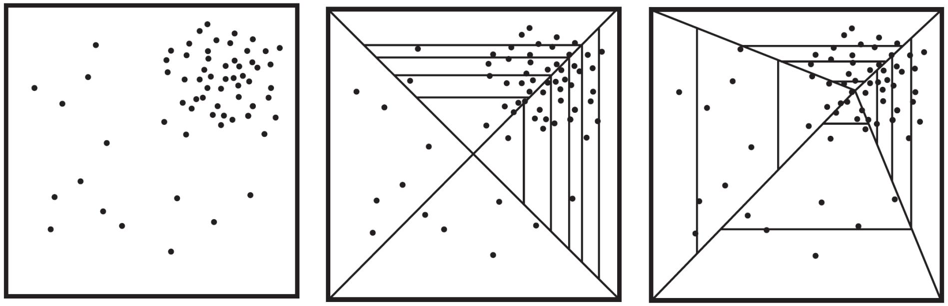


+ Pyramid Technique

- The effectiveness is sensitive to query position
  - Query region vs. search region



- Non-uniform distribution – design non-uniform pyramid



+ Outline

- Motivation
  - Examples
  - Why
  - What to expect
- Technique
  - X-Tree
  - Pyramid
  - VA-File
  - iDistance
  - Other techniques
- Summary

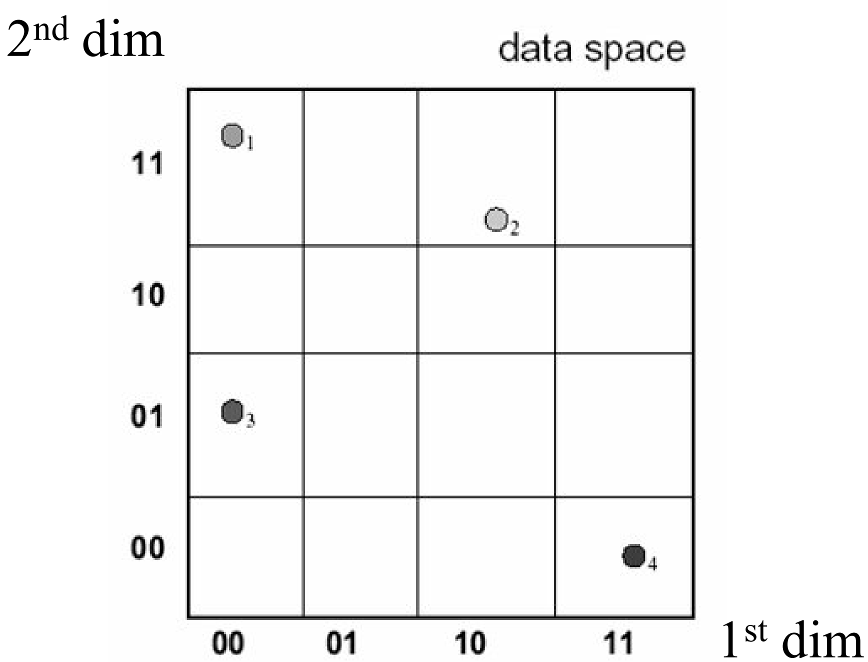
+ Vector Approximation (VA)-File

- In High-D spaces, tree-based indexing structures may examine a very large fraction of leave nodes
  - Since the MBRs heavily overlap if data are distributed uniformly
- Let go of the hierarchies and sequentially scan the whole data set
  - Sequential scan is much faster than random read due to the cost of disk seek operations
- Natural question: how to speed up linear scan?
  - Using approximation to compress vector data
  - Easing the computation and reducing the amount of data to exam during the search

[1] A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces, VLDB, 1998 VLDB Test-of-time award paper

+ Basic Idea

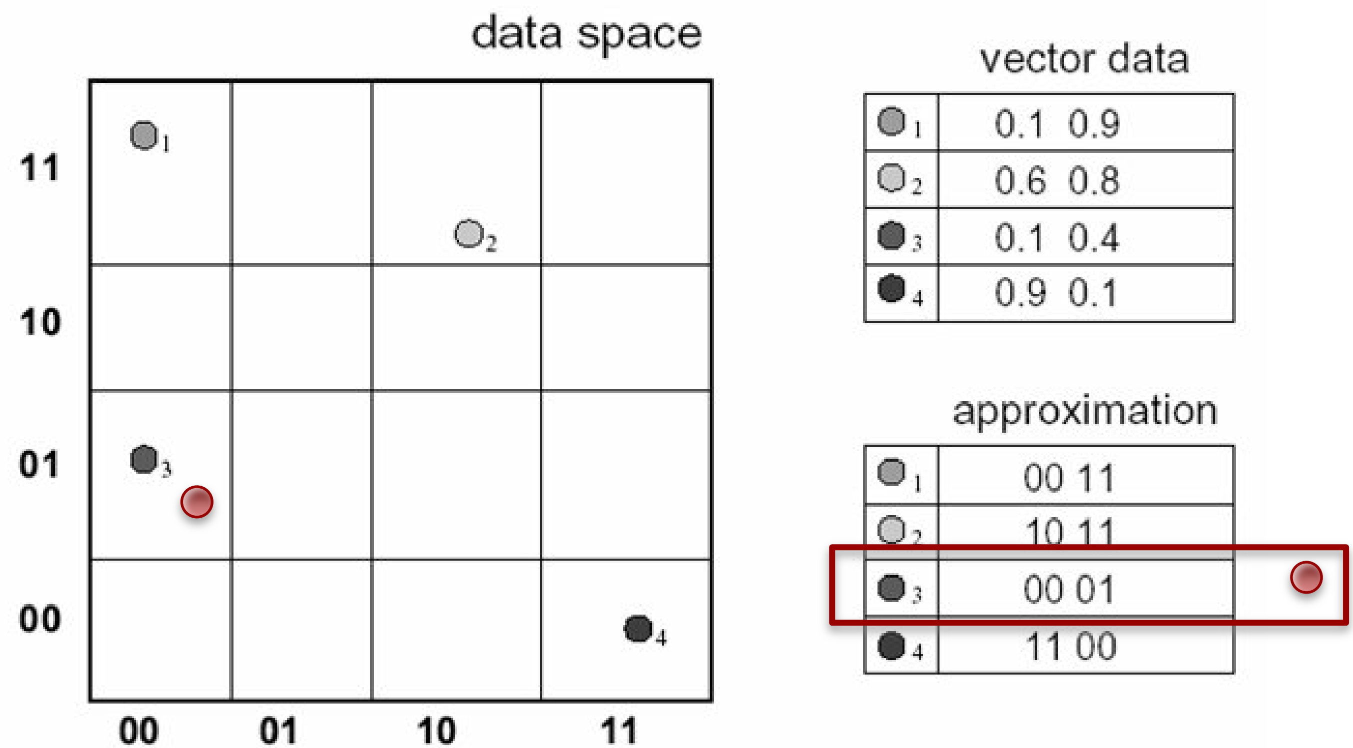
- For each dimension i, a small number of bits b_i is assigned to divide the dimension into 2^{b_i} intervals
- Let b be the sum of all dimensions' b_i's,
  - The data space is divided into 2^b cells
  - Every point is represented by b bits



- ✓ 1st dim: 2 bits
- ✓ 2nd dim: 2 bits
- ✓ In total $2^2 \times 2^2 = 2^{2+2} = 2^4 = 16$ cells

+ Building VA-file and Query

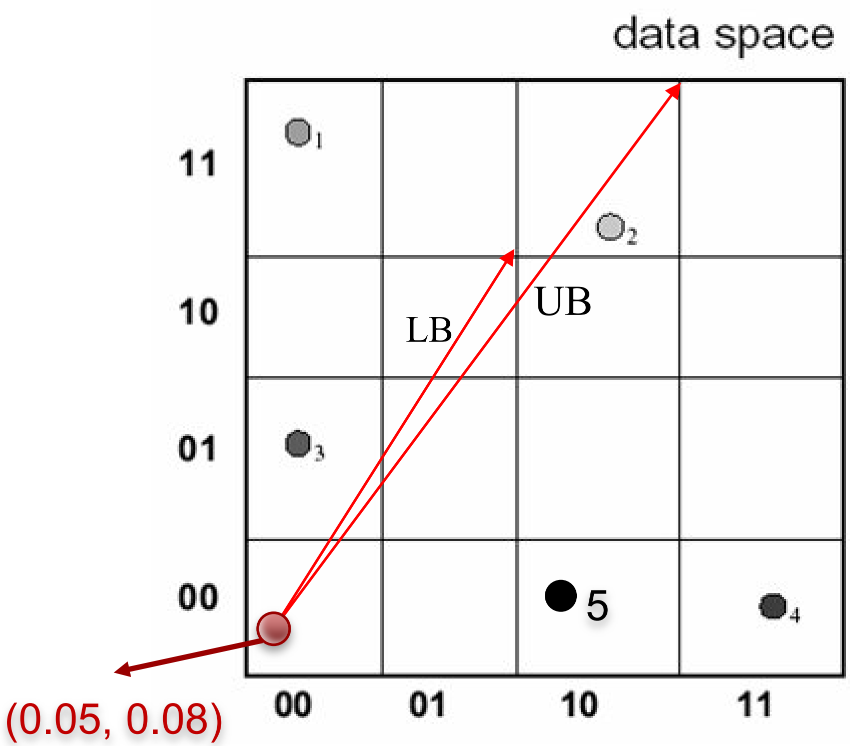
- Map objects to cells



- Each cell has a bit representation with length b
- The VA-file itself is simply an array of bits concatenation based on the quantization of the original feature vectors.

+ VA-file 2-Phase NN search

- **Phase 1: Filtering**
  - The VA-file is sequentially scanned.
  - The lower and upper bounds on the distance for each object's approximation (i.e., VA) is then computed.
  - Assume δ is the smallest upper bound so far, eliminate approximations with a lower bound that exceeds δ



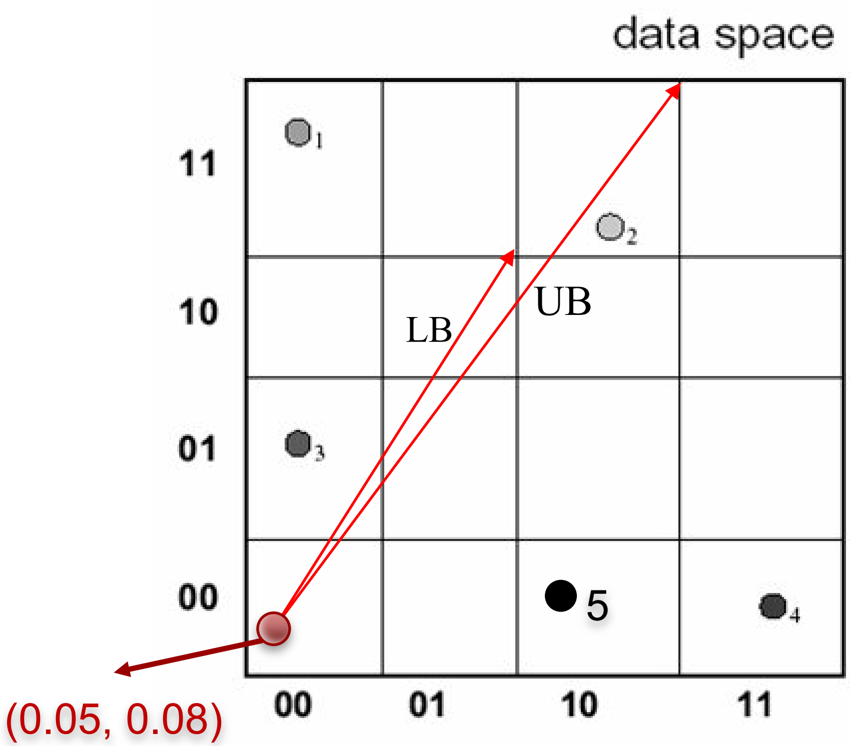
vector data

1	0.1 0.9
2	0.6 0.8
3	0.1 0.4
4	0.9 0.1

	approximation	LB, UB
1	00 11	0.67, 0.94
2	10 11	0.81, 1.16
3	00 01	0.17, $\boxed{0.47}$
4	11 00	0.70, 0.97
5	10 00	0.45, 0.72

+ VA-file 2-Phase NN search

- **Phase 1: Filtering**
  - The VA-file is sequentially scanned.
  - The lower and upper bounds on the distance for each object's approximation (i.e., VA) is then computed.
  - Assume δ is the smallest upper bound so far, eliminate approximations with a lower bound that exceeds δ



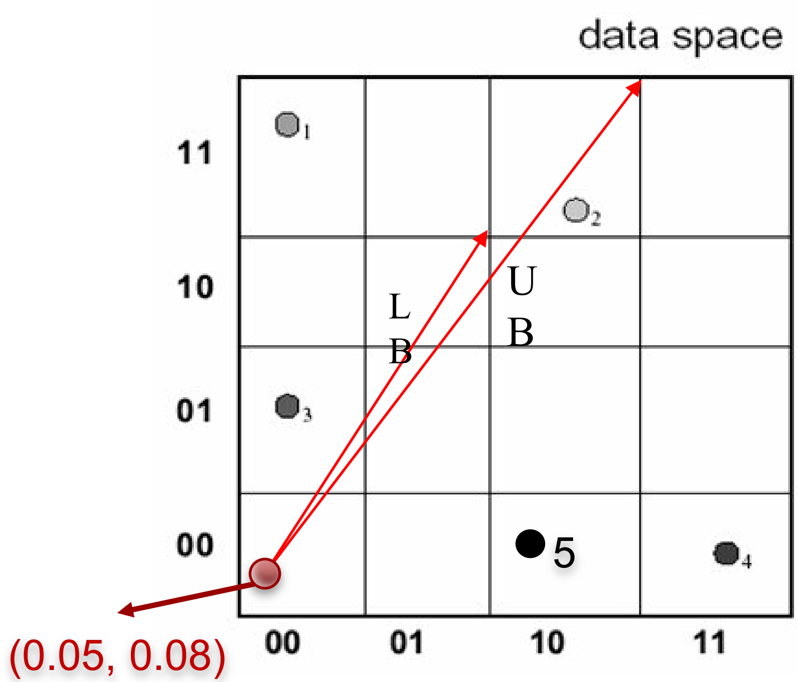
vector data

1	0.1 0.9
2	0.6 0.8
3	0.1 0.4
4	0.9 0.1

	approximation	LB, UB
1	00 11	~~0.67, 0.94~~
2	10 11	~~0.81, 1.16~~
3	00 01	0.17, $\boxed{0.47}$
4	11 00	~~0.70, 0.97~~
5	10 00	0.45, 0.72

+ VA-file 2-Phase NN search

- **Phase 2: Refine**
  - After the filtering step, a small set of candidates remain
  - Candidates are sorted in ascend order by lower bound P3, P5
  - Calculate the current NN distance by iteration through the sorted candidates
    - If a lower bound is encountered that exceeds the nearest distance seen so far, the VA-file method stops



vector data

1	0.1 0.9
2	0.6 0.8
3	0.1 0.4
4	0.9 0.1

approximation

		LB, UB	Dis.
1	00 11	~~0.67, 0.94~~	
2	10 11	~~0.81, 1.16~~	
3	00 01	0.17, $\boxed{0.47}$	0.32
4	11 00	~~0.70, 0.97~~	
5	10 00	0.45, 0.72	

+ What We can Learn from VA File?

- How to speed up linear scan ?

Answer: Use approximation!

- Use only b_i bits per dimension
  - Floating points: 32bits per dimension
  - Speed up the scan by a factor of $32/b_i$
- Identify all points which could be returned as an answer
- Verify the points by accessing the original feature vectors

+ Limits of VA-file

- The total cost of VA-file also includes the **random access (I/O)** to the candidates. If the candidate set is large, save in linear scan on VA-file will be offset.
- Large b_i:
  - Small number of candidates, but
  - Large VA-file
- VA-file performs best in **uniformly** distribution data. However, real data exhibit certain degree of skewness.
  - Assume the features are independent
    - Because the slices are obtained independently
  - VA-file divides the dimensions either for equal size or for equal population.

+ Outline

- Motivation
  - Examples
  - Why
  - What to expect
- Technique
  - X-Tree
  - Pyramid
  - VA-File
  - iDistance
  - Other techniques
- Summary

+ iDistance[1]

- iDistance is simple, but efficient
- It is a distance and partition based index
- It can be used for approximate search
- The index can be integrated to existing systems easily
- Another representative way to handle high dimensional data

[1] Indexing the Distance: An Efficient Method to KNN Search, VLDB, 2001

+ Basic Ideas

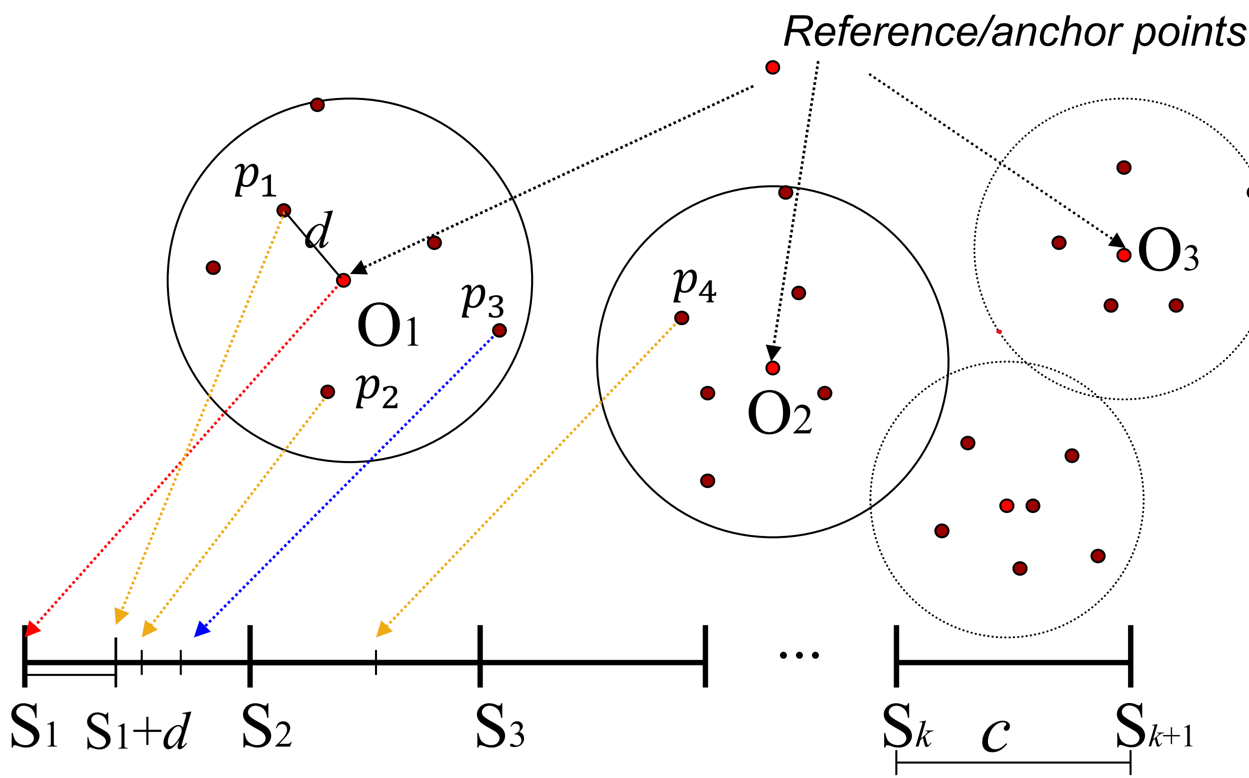
- Observations
  1. The similarity/dissimilarity between points can be derived with reference to a chose reference point
  2. Points can be ordered based on their distance to the reference point
  3. Distance is essentially a single dimensional value
     - Reuse the existing 1-D index like B+ -Tree

+ Basic Ideas

- Data points are partitioned into clusters/partitions
- Points are transformed into 1-D space
  1. The high-dimensional space is split into partitions
     - $P_0, P_1, \dots, P_{m-1}$
  2. A <u>reference point</u> is identified for each partition
     - $O_0, O_1, \dots, O_{m-1}$
  3. A data point $p(x_1, x_2, \dots, x_d)$ in the i^{th} partition can be referenced via O_i in terms of the distance from p to O_i
     - Index key y: $y = i \times c + dist(p, O_i)$
       - c: a constant of the data range
       - All points in partition P_i mapped to range $[i \times c, (i+1) \times c]$
- Data points are indexed based on similarity (metric distance) to such a point using a standard B^+-tree

+ Indexing Points Based on Similarity

Indexing points based on similarity



Assume: $Dist(p_1, O_1) = 5$, $Dist(p_4, O_2) = 5$
We set $c = 100$
$iDist(p_1) = 105$ and $iDist(p_4) = 205$

+ The range of search NN

- The triangle inequality

$$dist(O_i, q) - dist(p, q) \le dist(O_i, p) \le dist(O_i, q) + dist(p, q)$$

- Consider the points within a range around q containing p

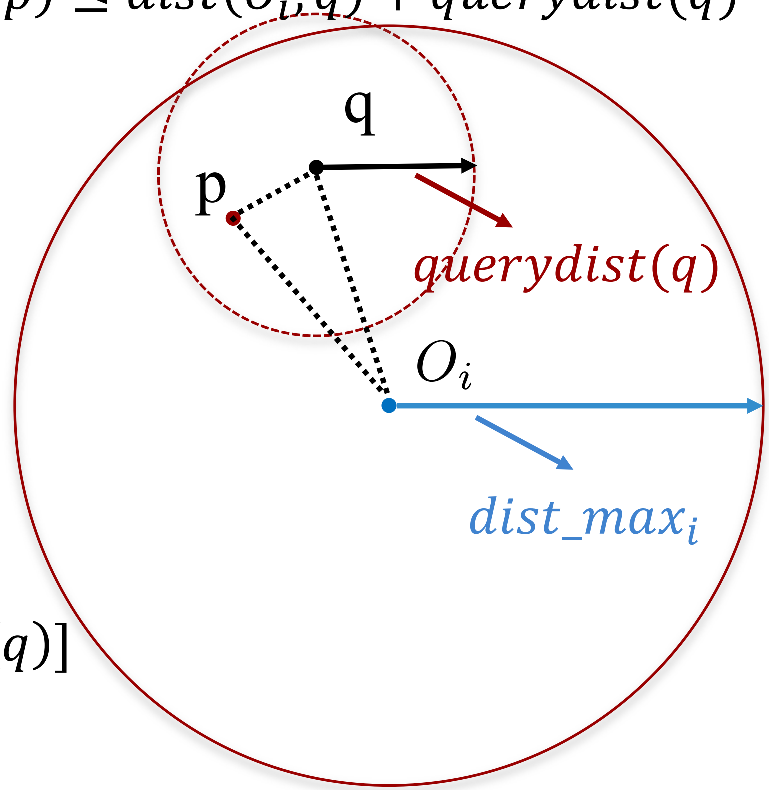
$$dist(O_i, q) - querydist(q) \le dist(O_i, p) \le dist(O_i, q) + querydist(q)$$

- The maximum distance range

$$dist(O_i, p) \le dist\_max_i$$

The final search space are points within the range

$$[dist(O_i, q) - querydist(q), \\ \min(dist\_max_i, dist(O_i, q) + querydist(q)]$$



+ The range of search NN

- The triangle inequality

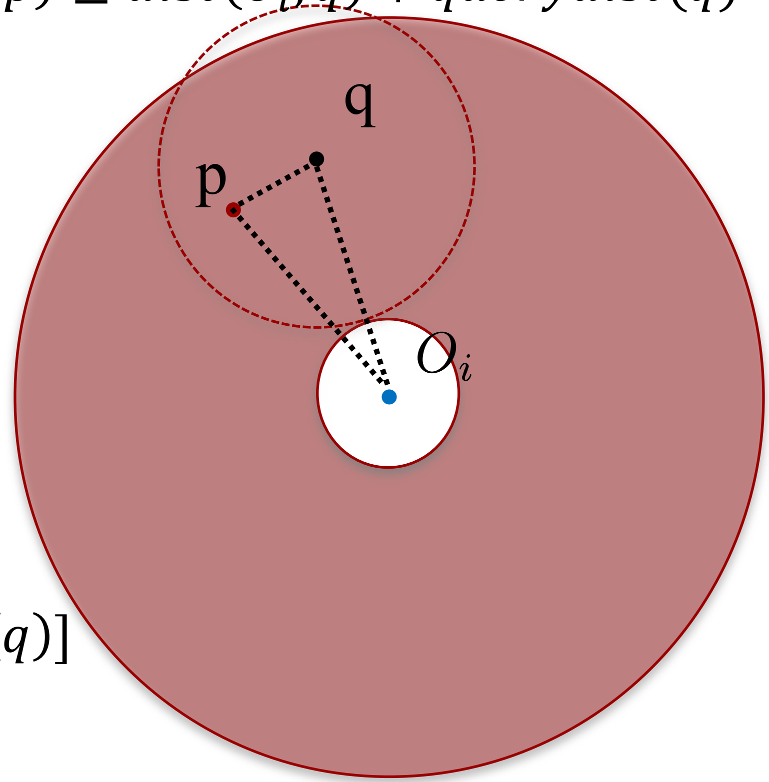
$$dist(O_i, q) - dist(p, q) \leq dist(O_i, p) \leq dist(O_i, q) + dist(p, q)$$

- Consider the points within a range around q containing p

$$dist(O_i, q) - querydist(q) \leq dist(O_i, p) \leq dist(O_i, q) + querydist(q)$$

- The maximum distance range

$$dist(O_i, p) \leq dist\_max_i$$



The final search space are points within the rage

$$[dist(O_i, q) - querydist(q),\\ \min(dist\_max_i, dist(O_i, q) + querydist(q)]$$

+ The range of search NN

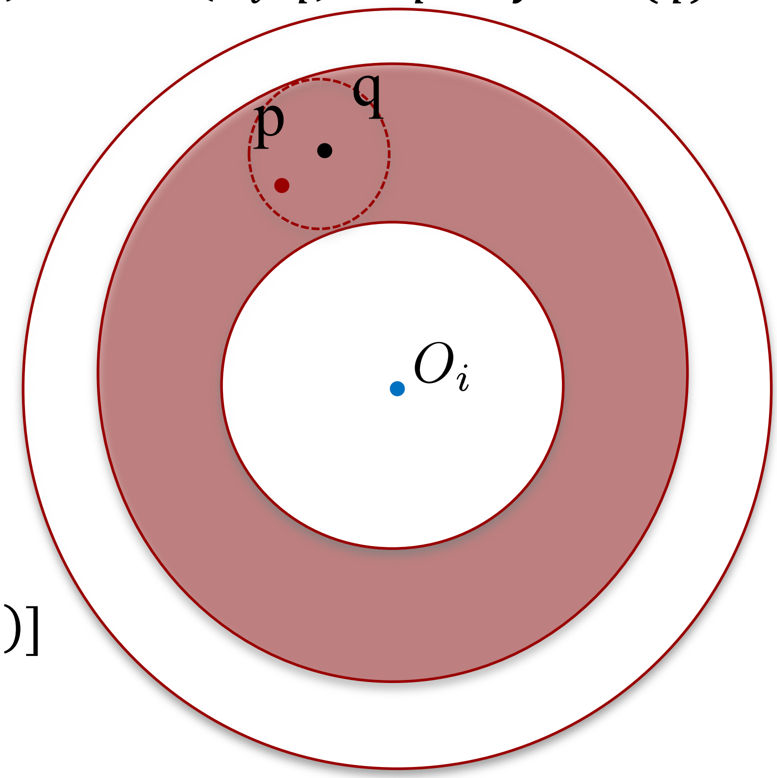
- The triangle inequality

$$dist(O_i, q) - dist(p, q) \leq dist(O_i, p) \leq dist(O_i, q) + dist(p, q)$$

- Consider the points within a range around q containing p

$$dist(O_i, q) - querydist(q) \leq dist(O_i, p) \leq dist(O_i, q) + querydist(q)$$

- The maximum distance range

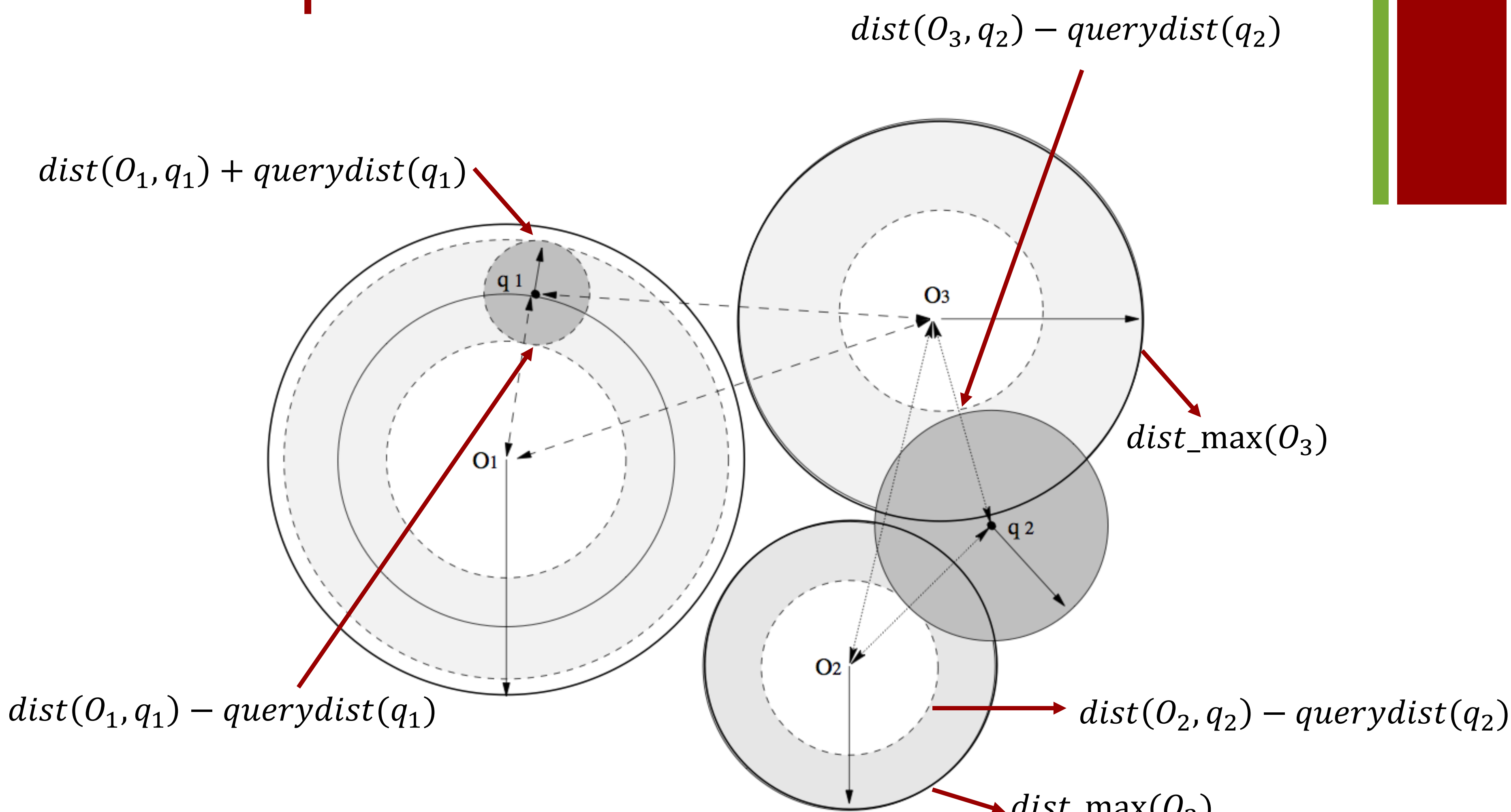


$$dist(O_i, p) \leq dist\_max_i$$

The final search space are points within the rage

$$[dist(O_i, q) - querydist(q), \\ \min(dist\_max_i, dist(O_i, q) + querydist(q)]$$

+ Example

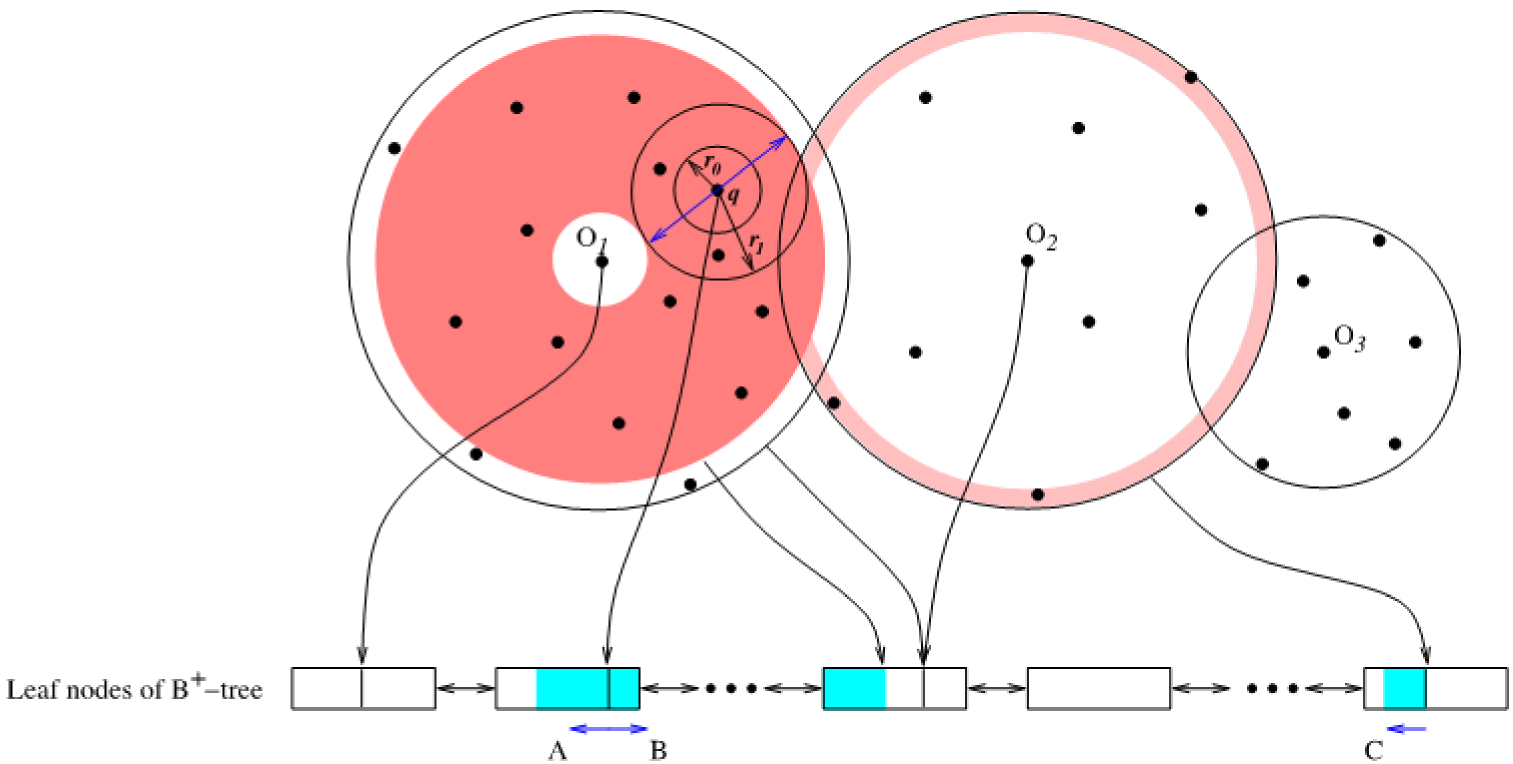


- Notice: all points along the same radius have the same value after transformation to distance
  - Points with the same values are not necessarily close to each other

+ kNN Search

- Query distance $r = querydist$ is enlarged incrementally
- For each query distance r, search is conducted around the query point q and overlapped partition until kNN is found
- For each partition, its **minimal** and **maximal** distance to the Reference Point are recorded in the auxiliary structure
  - The auxiliary structure: {(O, min, max)}
- Scanning the auxiliary structure to identify the partitions whose data space overlaps with the query sphere (*q*, *r*)

+ Graphical Illustration of kNN search



A range in B+-tree

✓ Searching region is enlarged until getting kNN

+ Formation of Clusters/Partitions

- **Equal Partitioning:**
  - Effective if data is uniformly distributed
  - However, data points are often non-uniformly distributed

- **Number of Clusters:**
  - Small number: More points are likely to have similar distance to a given reference point
  - Large number: More circles are likely to overlap and incurs additional traversal and searching

+ Outline

- Motivation
  - Examples
  - Why
  - What to expect
- Technique
  - VA-File
  - X-Tree
  - Pyramid
  - iDistance
  - Other techniques…
- Summary

+ List of techniques

- Hierarchical Tweaking
  - **X-Tree** [VLDB'96]
  - SS-Tree, SR-Tree, Ball-Tree
- Transformation based
  - **Pyramid Technique** [SIGMOD'98]
  - **iDistance** [VLDB'01]
  - Optimal one-dimensional distance B+ tree [SIGMOD'05]
- Dimensionality reduction
  - Local Dimensionality Reduction (LDR)
  - Multi-level Mahalanobis based Dimensionality Reduction (MMDR)
- Data compression based
  - Vector Approximation (**VA-File**) [VLDB'98]
- Hybrid of tree-like structure and data compression
  - Independent Quantization (IQ-tree)
  - Local Digital Coding (LDC)
- Approximate search
  - Locality Sensitive Hashing (LSH)
  - Vector Quantization (VQ-index)
  - Spatial Approximation Sample Hierarchy (SASH)

+ Outline

- Motivation
  - Examples
  - Why
  - What to expect
- Technique
  - VA-File
  - X-Tree
  - Pyramid
  - iDistance
  - Other techniques…
- Summary

+ Summary

- “Curse of Dimensionality” is a real problem which is very difficult to solve
- It is counter-intuitive that similarity search can be meaningless with more information captured
- It is also counter-intuitive that sophisticated indexing structures may not even be as efficient as a linear scan over the entire database
- People often use pivot-points based indexing and data compression to deal with high dimensional data
- This is still an open problem, with most solutions which are either data dependent or domain specific

+ Readings

- Christian Böhm, Stefan Berchtold and Daniel Keim: "Searching in High-dimensional Spaces: Index Structures for Improving the Performance of Multimedia Databases", *ACM Computing Surveys* 33 (3), 2001.
- Other papers on X-Tree, Pyramid Technique, VA-file and iDistance.

+ Advanced Techniques for High Dimensional Data

- ❑ Course Introduction
- ❑ Introduction to Spatial Databases
- ❑ Spatial Data Organization
- ❑ Spatial Query Processing
- ❑ Managing Spatiotemporal Data
- ❑ Managing High-Dimensional Data
- ❑ Introduction to Multimedia Database-next week
- ❑ Route Planning in Road Network
- ❑ When AI Meets High-Dimensional Data
- ❑ Trends and Course Review