

Multimedia Information Systems

Samson Cheung

EE 639, Fall 2004

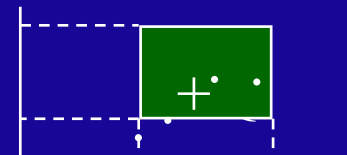
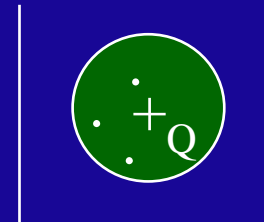
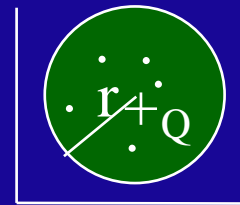
Lecture 19: Data structure for Similarity Search

What is similarity search?

- **Finding the record(s) in a large dataset closest to a given query**
- **Applications:**
 - **Content-based retrieval**
 - **Data-mining and Pattern recognition**
 - **Geographic information system (GIS)**
 - where is the closest post office?
- **Our focus is on efficient data structure (indexing) and the associated signal processing algorithms**
 - **Easy insertion and deletion**
 - **Fast query time**

Query Types

- ϵ -search: the distance tolerance is specified, used at the earlier stage and can be very “loose”.
- K-nearest-neighbor-queries: The user specifies the number of close matches to the given query point.
- Range queries: An interval is given for each dimension of the feature space and all the records which fall inside this hypercube are retrieved.



Binary search for 1-D feature

12.3 45 -3 5.333 6 100 9.4 10 3.4 72.56

Sort

-3	3.4	5.333	6	9.4	10	12.3	45	72.56	100
----	-----	-------	---	-----	----	------	----	-------	-----

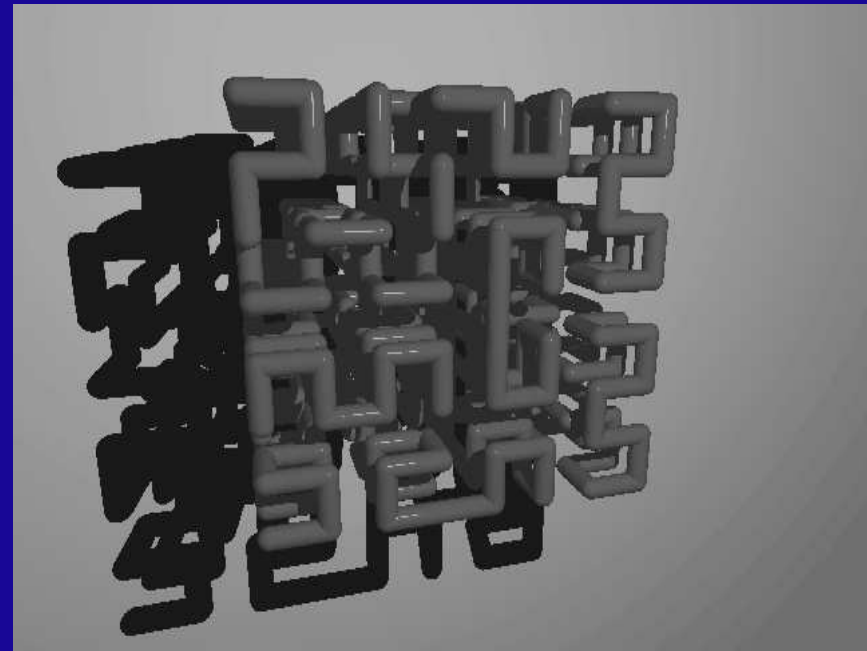
Search

23.4

-3	3.4	5.333	6	9.4	10	12.3	45	72.56	100
----	-----	-------	---	-----	----	------	----	-------	-----

$O(\log N)$
- Much better
than sequential
search !!

Better rasterization – space filling curve



- **Space-filling curve**
 - Still have problems at corners
 - Get worse for higher-dimension (more corners)

Multidimensional Index Structures

- Partitioning in the higher dimensional space
- Some common structures:
 - k-d Trees
 - Point quad-tree
 - MX quad-tree
 - R Trees, R*, TV, SS
 - and many others

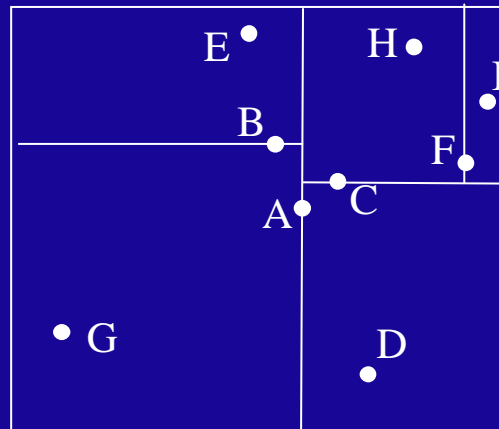
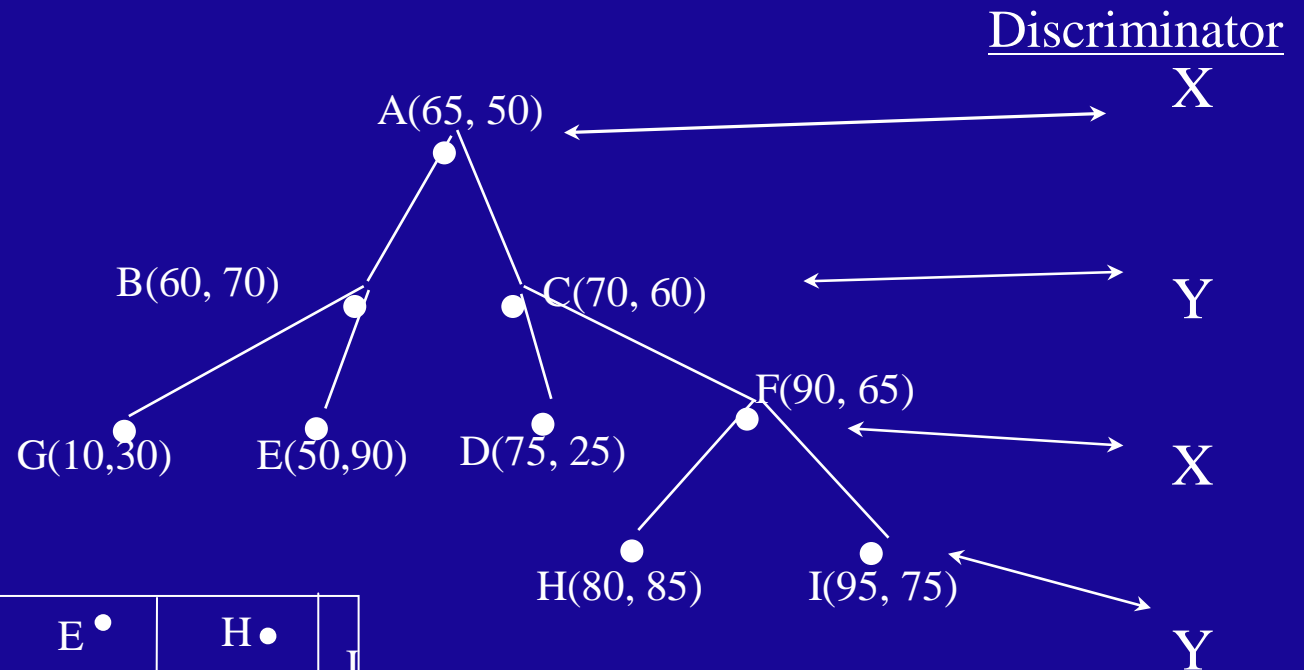
k-d Trees

- **k-d tree is a multidimensional binary search tree.**
- **Each node consists of a “record” and two pointers. The pointers are either null or point to another node.**
- **Nodes have levels and each level of the tree discriminates for one attribute.**
- **The partitioning of the space with respect to various attributes alternates between the various attributes of the n-dimensional search space.**

K-d tree example

Input Sequence

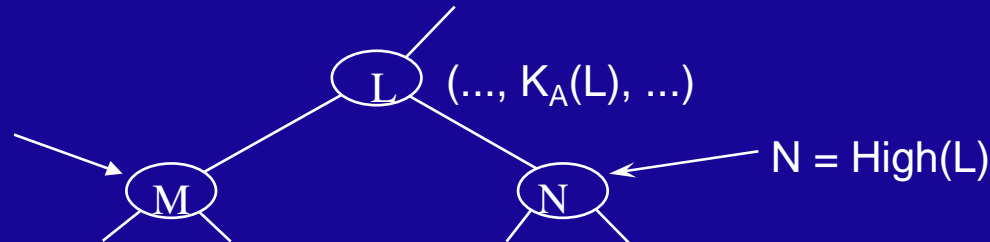
- A = (65, 50)
- B = (60, 70)
- C = (70, 60)
- D = (75, 25)
- E = (50, 90)
- F = (90, 65)
- G = (10, 30)
- H = (80, 85)
- I = (95, 75)



k-d Tree: Search Algorithm

- Notations:

$M = \text{Low}(L)$



$\text{Disc}(L)$: The discriminator at L's level

$K_A(L)$: The A-attribute value of L

$\text{Low}(L)$: The left child of L

$\text{High}(L)$: The right child of L

- Algorithm: Search for $P(K_1, \dots, K_n)$

$Q := \text{Root};$ /* Q will be used to navigate the tree */

While NOT DONE DO the following:

if $K_i(P) = K_i(Q)$ for $i = 1, \dots, n$ then we have
located the node and we are DONE

Otherwise if $A = \text{Disc}(Q)$ and $K_A(P) < K_A(Q)$

then $Q := \text{Low}(Q)$

else $Q := \text{High}(Q)$

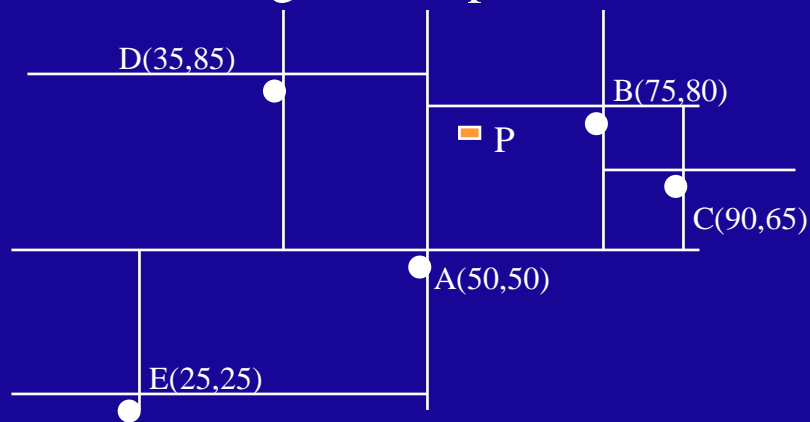
Point-Quad Trees

- **Why limit to binary partition?**
- **Each node of a k -dimensional quad tree partitions the object space into k quadrants.**
- **The partitioning is performed along all search dimensions and is data dependent, like k -d trees.**

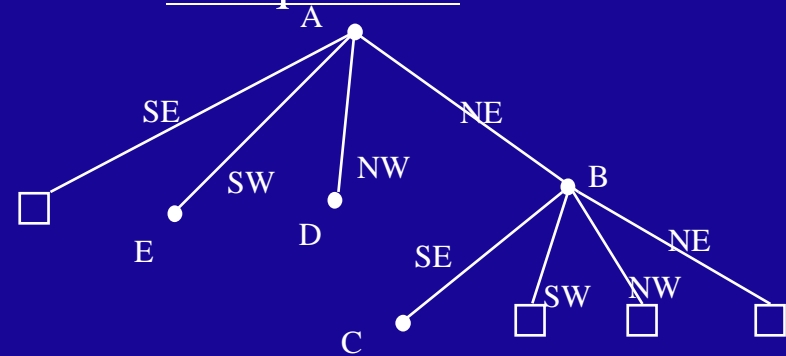
Point-Quad Tree Example

Example:

Partitioning of the space



The quad tree



To find/insert $P(55, 75)$:

- Since $X_A < X_P$ and $Y_A < Y_P \Rightarrow$ go to NE (i.e., B).
- Since $X_B > X_P$ and $Y_B > Y_P \Rightarrow$ go to SW, which in this case is null.

MX-quadtree

- Both k-d tree and point-quad tree may divide the space unevenly due to the uneven distribution of the data points
- MX-quadtree, are similar to point quadtree tree except that they divide the embedding space.
- Each split evenly divides a region

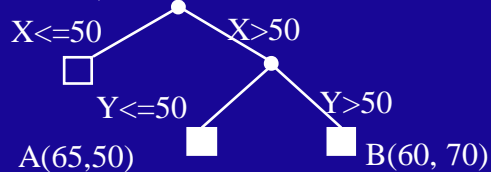
MX-Quad tree Example

Example: Construction of a MX-quad tree

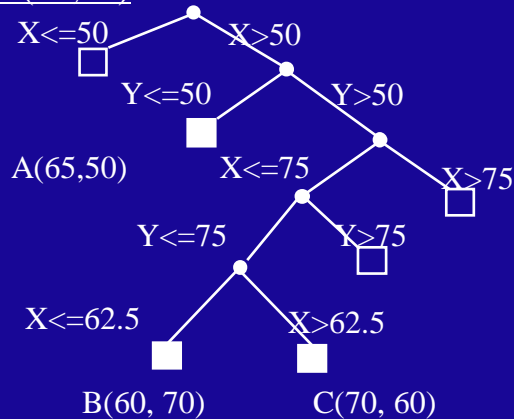
Insert A(65,50):



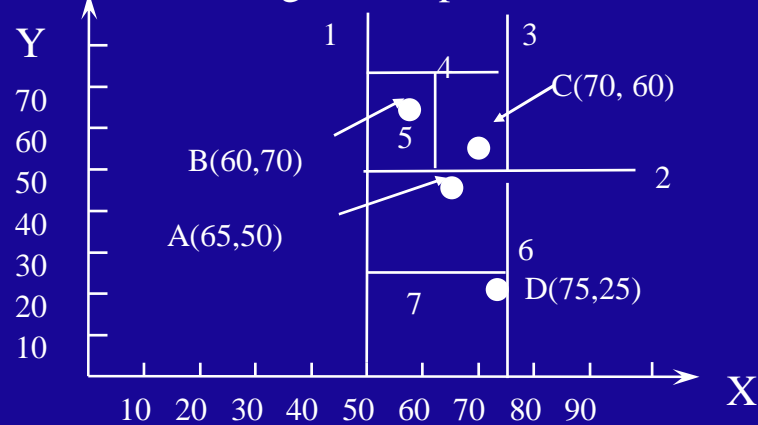
Insert B(60,70):



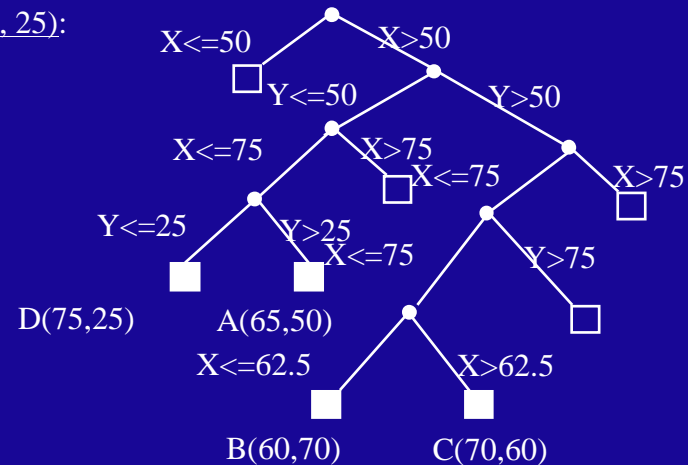
Insert C(70,60):



Partitioning of the space



Insert D(75,25):

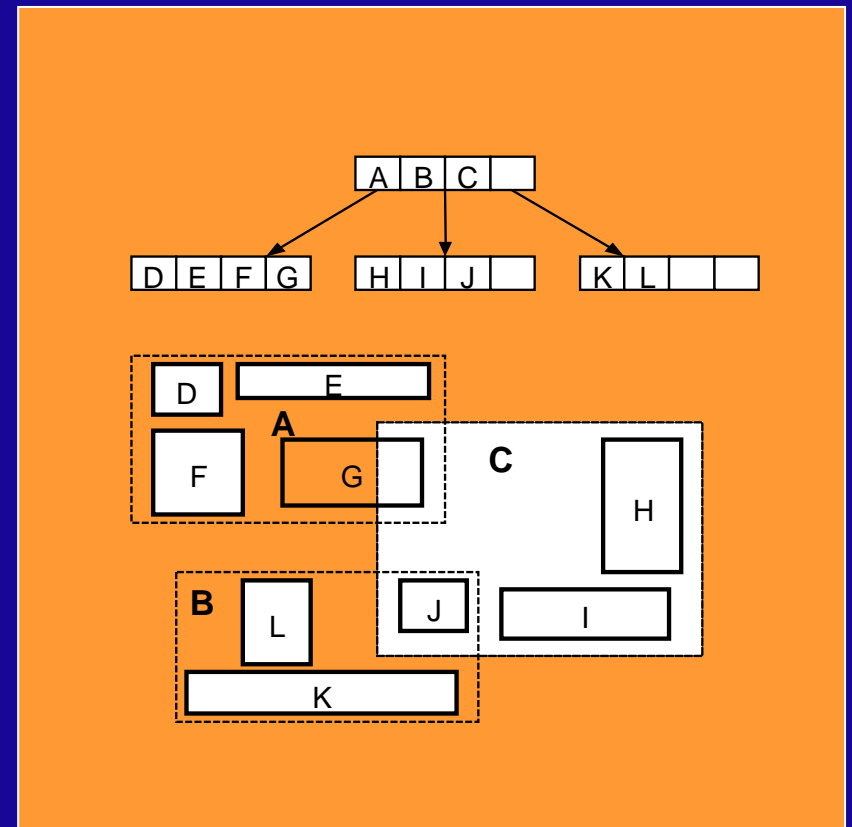


R-tree

- All previous structures are not suitable for disk access because of the random access of each data object
- Disk typically organize blocks of data called pages – much more efficient to load one page at a time
- R-tree (similar to B-tree in 1-D)
 - Main memory : store summary of data
 - Data are stored at leaf nodes.
 - Each leaf node contain multiple data objects

R-tree continued

- Root and intermediate nodes corresponds to the smallest rectangle that encloses its child nodes
- Each node must be at least half full – minimize the height of the tree
- Leaf nodes contain pointers to the actual objects
- A rectangle may be spatially contained in several nodes (e.g., *J*), yet it can be associated with only one node.
 - Need to search multiple branches!!

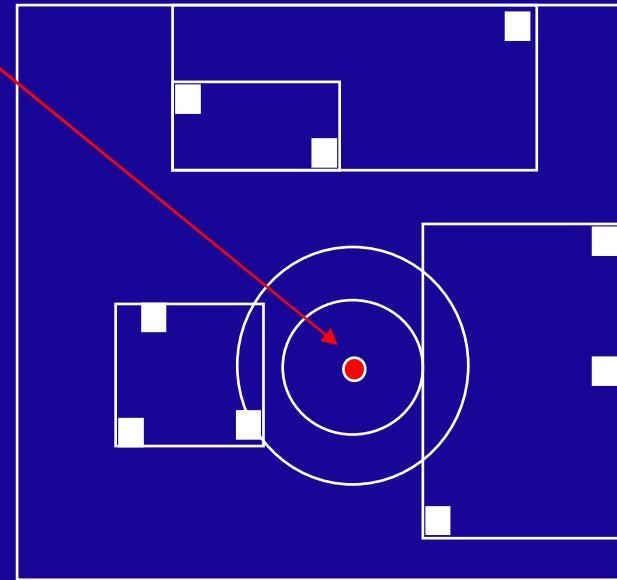


Low-dimensional case

■ R-tree

- Keep Number of Bounding Boxes small (Storage)
- Each BB should have roughly the same number of points.

NN search



High-dimensional cases

Assume $[0,1]^d$; N uniform dist. data pts; M BBs.

- For fixed N , $E(\text{nn-dist}) = O(\sqrt{d})$
 - Length of main diagonal of $[0,1]^d = \sqrt{d}$.
- If BB is a d -cube(s), $V = s^d$.
 - $d = 100, s = 0.5, V = 8e-31 \Rightarrow M \cong N$
- To keep M small, lower individual BB to dim $d' = O(\log_2 N/M) \Rightarrow V(d'\text{-cube}(0.5)) = M/N$
 - $l_{\max} = \text{max. distance from BB} = 0.5\sqrt{\log_2(N/M)}$
- $d > \log N, l_{\max} < E(\text{nn-dist}) \Rightarrow \text{Sequential search!!}$

Curse of dimensionality

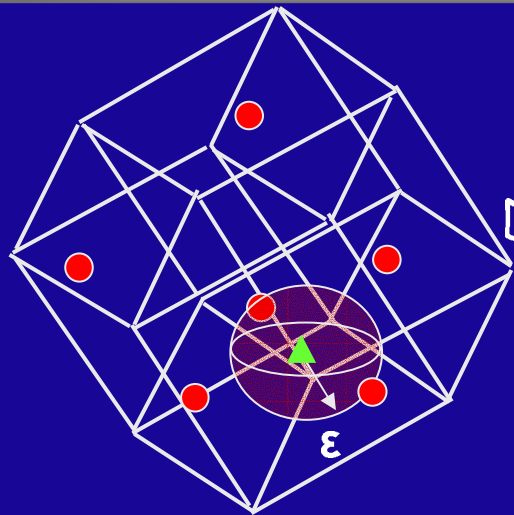
Similarity search on high-dimensional vectors is hard !

Typical indexing techniques reduce to sequential search for 10 or higher dimensions .

[Weber, Schek, Blott98]

- Average distance between closest neighbors increase with dimension
- # bounding regions grows exponentially with dimension
- Volume of each region shrinks exponentially with dimension

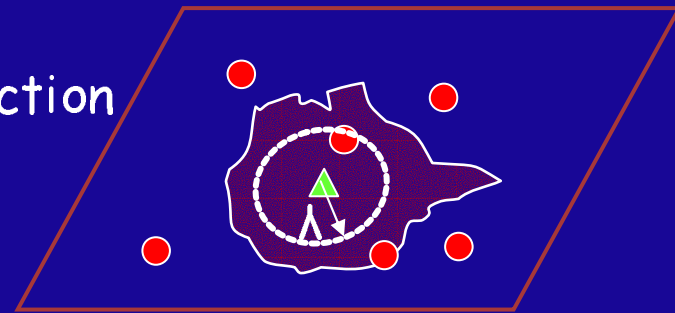
Dimension reduction for fast search



High dimensional space
 $d(x,y)$



Dimension Reduction
Mapping T



Low dimensional space
 $d'(T(x),T(y)) + \text{Indexing}$

Goal: Need to find mapping T and metric $d'(\bullet, \bullet)$ that can preserve distance relationship

Drawback: May not be able to find all similar objects (approximate similarity search)