Techniques for Similarity Searching in Multimedia Databases

Hanan Samet

hjs@cs.umd.edu

Department of Computer Science

University of Maryland

College Park, MD 20742, USA

Part A: Overview

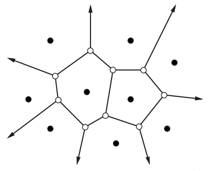
1. Similarity searching
2. Voronoi diagrams
3. Approximate Voronoi diagrams
4. Curse of dimensionality

Similarity Searching

- Important task when trying to find patterns in applications involving mining different types of data such as images, video, time series, text documents, DNA sequences, etc.
- Similarity searching module is a central component of content-based retrieval in multimedia databases
- Problem: finding objects in a data set S that are similar to a query object q based on some distance measure d which is usually a distance metric
- Sample queries:
  1. point: objects having particular feature values
  2. range: objects whose feature values fall within a given range or where the distance from some query object falls into a certain range
  3. nearest neighbor: objects whose features have values similar to those of a given query object or set of query objects
  4. closest pairs: pairs of objects from the same set or different sets which are sufficiently similar to each other (variant of spatial join)
- Responses invariably use some variant of nearest neighbor finding

Voronoi Diagrams

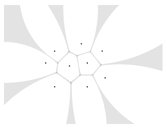
- Apparently straightforward solution:
  1. Partition space into regions where all points in the region are closer to the region's data point than to any other data point
  2. Locate the Voronoi region corresponding to the query point



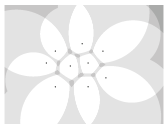
- Problem: storage and construction cost for N d-dimensional points is $\Theta(N^{d/2})$
- Impractical unless resort to some high-dimensional approximation of a Voronoi diagram (e.g., OS-tree) which results in approximate nearest neighbors
- Exponential factor corresponding to the dimension d of the underlying space in the complexity bounds when using approximations of Voronoi diagrams (e.g., (t,ϵ)-AVD) is shifted to be in terms of the error threshold ϵ rather than in terms of the number of objects N in the underlying space
  1. $(1,\epsilon)$-AVD: $O(N/\epsilon^{d-1})$ space and $O(\log(N/\epsilon^{d-1}))$ time for nearest neighbor query
  2. $(1/\epsilon^{(d-1)2},\epsilon)$-AVD: $O(N)$ space and $O(t+\log N)$ time for nearest neighbor query

Approximate Voronoi Diagrams (AVD)

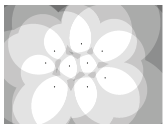
- Example partitions of space induced by ϵ neighbor sets
- Darkness of shading indicates cardinality of nearest neighbor sets with white corresponding to 1



$(\epsilon = 0.10)$

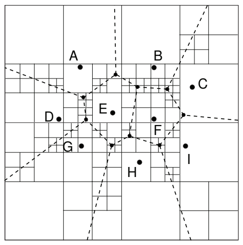


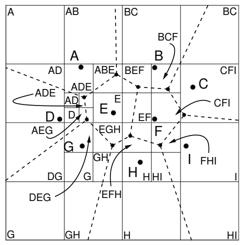
$(\epsilon = 0.30)$



$(\epsilon = 0.50)$

Approximate Voronoi Diagrams (AVD) Representations





- Partition underlying domain so that for $\epsilon \geq 0$, every block b is associated with some element r_b in S such that r_b is an ϵ-nearest neighbor for all of the points in b (e.g., AVD or (1,0.25)-AVD)
- Allow up to $t \geq 1$ elements $r_{ib}(1 \leq i \leq t)$ of S to be associated with each block b for a given ϵ, where each point in b has one of the r_{ib} as its ϵ-nearest neighbor (e.g., (3,0)-AVD)

Problem: Curse of Dimensionality

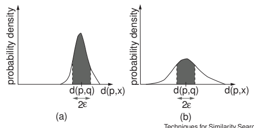
- Number of samples needed to estimate an arbitrary function with a given level of accuracy grows exponentially with the number of variables (i.e., dimensions) that comprise it (Bellman)
- For similarity searching, curse means that the number of points in the data set that need to be examined in deriving the estimate ($\equiv$ nearest neighbor) grows exponentially with the underlying dimension
- Effect on nearest neighbor finding is that the process may not be meaningful in high dimensions
- When ratio of variance of distances and expected distances, between two random points p and q drawn from the data and query distributions, converges to zero as dimension d gets very large (Beyer et al.)

$$\lim_{d\to\infty} \frac{\mathrm{Variance}[dist(p,q)]}{\mathrm{Expected}[dist(p,q)]} = 0$$

  1. distance to the nearest neighbor and distance to the farthest neighbor tend to converge as the dimension increases
  2. implies that nearest neighbor searching is inefficient as difficult to differentiate nearest neighbor from other objects
  3. assumes uniformly distributed data
- Partly alleviated by fact that real-world data is rarely uniformly-distributed

Alternative View of Curse of Dimensionality

- Probability density function (analogous to histogram) of the distances of the objects is more concentrated and has a larger mean value
- Implies similarity search algorithms need to do more work
- Worst case when $d(x,x)=0$ and $d(x,y)=1$ for all $y \neq x$
- Implies must compare every object with every other object
  1. can't always use triangle inequality to prune objects from consideration
  2. triangle inequality (i.e., $d(q,p) \leq d(p,x)+d(q,x)$) implies that any x such that $|d(q,p)-d(p,x)| > \epsilon$ cannot be at a distance of ϵ or less from q as $d(q,x) \geq d(q,p)-d(p,x) > \epsilon$
  3. when ϵ is small while probability density function is large at $d(p,q)$, then probability of eliminating an object from consideration via use of triangle inequality is remaining area under curve which is small (see left) in contrast to case when distances are more uniform (see right)



Other Problems

- Point and range queries are less complex than nearest neighbor queries
  1. easy to do with multi-dimensional index as just need comparison tests
  2. nearest neighbor require computation of distance
     - Euclidean distance needs d multiplications and $d-1$ additions
- Often we don't know features describing the objects and thus need aid of domain experts to identify them

Solutions Based on Indexing

1. Map objects to a low-dimensional vector space which is then indexed using one of a number of different data structures such as k-d trees, R-trees, quadtrees, etc.
   - use dimensionality reduction: representative points, SVD, DFT, etc.
2. Directly index the objects based on distances from a subset of the objects making use of data structures such as the vp-tree, M-tree, etc.
   - useful when only have a distance function indicating similarity (or dis-similarity) between all pairs of N objects
   - if change distance metric, then need to rebuild index — not so for multidimensional index
3. If only have distance information available, then embed the data objects in a vector space so that the distances of the embedded objects as measured by the distance metric in the embedding space approximate the actual distances
   - commonly known embedding methods include multidimensional scaling (MDS), Lipschitz embeddings, FastMap, etc.
   - once a satisfactory embedding has been obtained, the actual search is facilitated by making use of conventional indexing methods, perhaps coupled with dimensionality reduction

Part B:Indexing Low and High Dimensional Spaces

1. Quadtree variants
2. k-d tree
3. R-tree
4. X-tree
5. Bounding sphere methods
   - SS-tree
   - SR-tree
6. Methods based on Voronoi diagrams
7. Pyramid technique
8. Methods based on a sequential scan

Simple Non-Hierarchical Data Structures

Sequential list

Name	X	Y
Chicago	35	42
Mobile	52	10
Toronto	62	77
Buffalo	82	65
Denver	5	45
Omaha	27	35
Atlanta	85	15
Miami	90	5

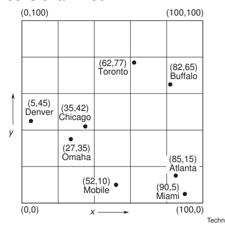
Inverted List

X	Y
Denver	Miami
Omaha	Mobile
Chicago	Atlanta
Mobile	Omaha
Toronto	Chicago
Buffalo	Denver
Atlanta	Buffalo
Miami	Toronto

Inverted lists:
1. 2 sorted lists
2. data is pointers
3. enables pruning the search with respect to one key

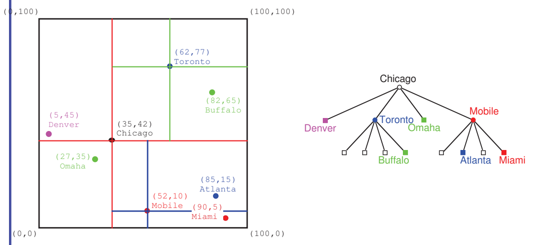
Grid Method

- Divide space into squares of width equal to the search region
- Each cell contains a list of all points within it
- Assume L_∞ distance metric (i.e., Chessboard)
- Assume C = uniform distribution of points per cell
- Average search time for k-dimensional space is $O(F \cdot 2k)$
  - F = number of records found = C, since query region has the width of a cell
  - 2^k = number of cells examined



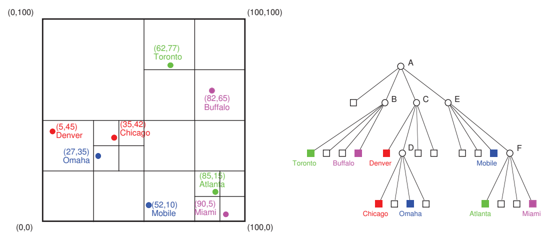
Point Quadtree (Finkel/Bentley)

- Marriage between uniform grid and a binary search tree



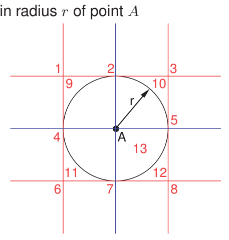
PR Quadtree

1. Regular decomposition point representation
2. Decompose whenever a block contains more than one point
3. Maximum level of decomposition depends on minimum point separation
   - if two points are very close, then decomposition can be very deep
   - can be overcome by viewing blocks as buckets with capacity c and only decomposing a block when it contains more than c points



Region Search

- Ex: Find all points within radius r of point A

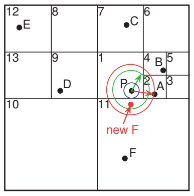


- Use of quadtree results in pruning the search space
- If a quadrant subdivision point p lies in a region l, then search the quadrants of p specified by l

1.	SE	5.	SW, NW	9.	All but NW	13.	All
2.	SE, SW	6.	NE	10.	All but NE		
3.	SW	7.	NE, NW	11.	All but SW		
4.	SE, NE	8.	NW	12.	All but SE		

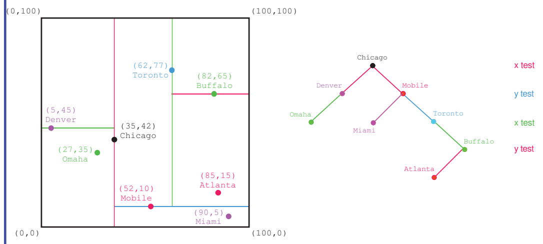
Finding Nearest Object

- Ex: find the nearest object to P
- Assume PR quadtree for points (i.e., at most one point per block)
- Search neighbors of block 1 in counterclockwise order
- Points are sorted with respect to the space they occupy which enables pruning the search space
- Algorithm:
  1. start at block 2 and compute distance to P from A
  2. ignore block 3, even if nonempty, as A is closer to P than any point in 3
  3. examine block 4 as distance to SW corner is shorter than the distance from P to A; however, reject B as it is further from P than A
  4. ignore blocks 6, 7, 8, 9, and 10 as the minimum distance to them from P is greater than the distance from P to A
  5. examine block 11 as the distance from P to the S border of 1 is shorter than distance from P to A; but, reject F as it is further from P than A
- If F was moved, a better order would have started with block 11, the southern neighbor of 1, as it is closest to the new F



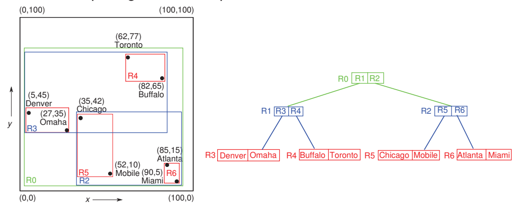
k-d tree (Bentley)

- Test one attribute at a time instead of all simultaneously as in the point quadtree
- Usually cycle through all the attributes
- Shape of the tree depends on the order in which the data is encountered



Minimum Bounding Rectangles: R-tree (Guttman)

- Objects grouped into hierarchies, stored in a structure similar to a B-tree
- Object has single bounding rectangle, yet area that it spans may be included in several bounding rectangles
- Drawback: not a disjoint decomposition of space (e.g., Chicago in R1+R2)
- Order (m, M) R-tree
  1. between $m \le M/2$ and M entries in each node except root
  2. at least 2 entries in root unless a leaf node
- X-tree (Berchtold/Keim/Kriegel): if split creates too much overlap, then instead of splitting, create a supernode

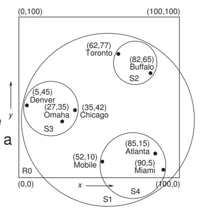


R*-tree (Beckmann et al.)

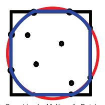
- Goal: minimize overlap for leaf nodes and area increase for nonleaf nodes
- Changes from R-tree:
  1. insert into leaf node p for which resulting bounding box has minimum increase in overlap with bounding boxes of p's brothers
     - compare with R-tree where insert into leaf node for which increase in area is a minimum (minimizes coverage)
  2. in case of overflow in p, instead of splitting p as in R-tree, reinsert a fraction of objects in p (e.g., farthest from centroid)
     - known as 'forced reinsertion' and similar to 'deferred splitting' or 'rotation' in B-trees
  3. in case of true overflow, use a two-stage process (goal: low coverage)
     - determine axis along which the split takes place
       a. sort bounding boxes for each axis on low/high edge to get $2d$ lists for d-dimensional data
       b. choose axis yielding lowest sum of perimeters for splits based on sorted orders
     - determine position of split
       a. position where overlap between two nodes is minimized
       b. resolve ties by minimizing total area of bounding boxes
- Works very well but takes time due to forced reinsertion

Minimum Bounding Hyperspheres

- SS-tree (White/Jain)
  1. make use of hierarchy of minimum bounding hyperspheres
  2. based on observation that hierarchy of minimum bounding hyperspheres is more suitable for hyperspherical query regions
  3. specifying a minimum bounding hypersphere requires slightly over one half the storage for a minimum bounding hyperrectangle
     - enables greater fanout at each node resulting in shallower trees
  4. drawback over minimum bounding hyperrectangles is that it is impossible cover space with minimum bounding hyperspheres without some overlap



- SR-tree (Katayama/Satoh)
  1. bounding region is intersection of minimum bounding hyperrectangle and minimum bounding hypersphere
  2. motivated by desire to improve performance of SS-tree by reducing volume of minimum bounding boxes

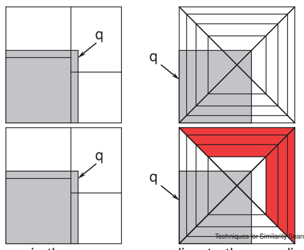


Avoiding Overlapping All of the Leaf Blocks

- Assume uniformly-distributed data
  1. most data points lie near the boundary of the space that is being split
     - Ex: for $d = 20$, 98.5% of the points lie within 10% of the surface
     - Ex: for $d = 100$, 98.3% of the points lie within 2% of the surface
  2. rarely will all of the dimensions be split even once
     - Ex: assuming at least $M/2$ points per leaf node blocks, and at least one split along each dimension, then total number of points N must be at least $2^d M/2$
     - if $d = 20$ and $M = 10$, then N must be at least 5 million to split along all dimensions once
  3. if each region is split at most once, and without loss of generality, split is in half, then query region usually intersects all the leaf node blocks
     - query selectivity of 0.01% for $d = 20$ leads to 'side length of query region'=0.63 which means that it intersects all the leaf node blocks
     - implies a range query will visit each leaf node block
- One solution: use a 3-way split along each dimension into three parts of proportion r, $1 - 2r$, and r
- Sequential scan may be cheaper than using an index due to high dimensions
  - We assume our data is not of such high dimensionality!

Pyramid Technique (Berchtold/Böhm/Kriegel)

- Subdivide data space as if it is an onion by peeling off hypervolumes that are close to the boundary
- Subdivide hypercube into $2d$ pyramids having the center of the data space as the tip of their cones
- Each of the pyramids has one of the faces of the hypercube as its base
- Each pyramid is decomposed into slices parallel to its base
- Useful when query region side length is greater than half the width of the data space as won't have to visit all leaf node blocks



- Pyramid containing q is the one corresponding to the coordinate i whose

Methods Based on a Sequential Scan

1. If neighbor finding in high dimensions must access every disk page at random, then a linear scan may be more efficient
   - advantage of sequential scan over hierarchical indexing methods is that actual I/O cost is reduced by being able to scan the data sequentially instead of at random as only need one disk seek
2. VA-file (Weber et al.)
   - use b_i bits per feature i to approximate feature
   - impose a d dimensional grid with $b = \sum_{i=1}^{d} b_i$ grid cells
   - sequentially scan all grid cells as a filter step to determine possible candidates which are then checked in their entirety via a disk access
   - VA-file is an additional representation in the form of a grid which is imposed on the original data
3. Other methods apply more intelligent quantization processes
   - VA+-file (Ferhatosmanoglu et al): decorrelate the data with KLT yielding new features and vary number of bits as well as use clustering to determine the region partitions
   - IQ-tree (Berchtold et al): hierarchical like an R-tree with unordered minimum bounding rectangles

Part C: Distance-Based Indexing

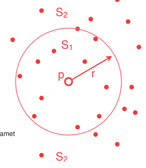
1. Distance
2. Ball partitioning methods
   - vp-tree
   - mvp-tree
3. General hyperplane partitioning methods
   - gh-tree
   - GNAT
   - mb-tree
4. M-tree
5. Sa-tree
6. kNN graph
7. Delaunay graph
8. Spatial approximation sample hierarchy (SASH)

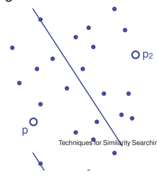
Basic Definitions

1. Often only information available is a distance function indicating degree of similarity (or dis-similarity) between all pairs of N data objects
2. Distance metric d: objects must reside in finite metric space (S, d) where for o_1, o_2, o_3 in S, d must satisfy
   - $d(o_1, o_2) = d(o_2, o_1)$ (symmetry)
   - $d(o_1, o_2) \geq 0$, $d(o_1, o_2) = 0$ iff $o_1 = o_2$ (non-negativity)
   - $d(o_1, o_3) \leq d(o_1, o_2) + d(o_2, o_3)$ (triangle inequality)
3. Triangle inequality is a key property for pruning search space
4. Non-negativity property enables ignoring negative values in derivations

Pivots

- Identify a distinguished object or subset of the objects termed *pivots* or *vantage points*
  1. sort remaining objects based on distances from pivots and build index
  2. use index to achieve pruning of other objects during search
- Given pivot $p \in S$, for all objects $o \in S' \subseteq S$, we know:
  1. exact value of $d(p, o)$,
  2. $d(p, o)$ lies within range $[r_{\mathrm{lo}}, r_{\mathrm{hi}}]$ of values (ball partitioning) (ball partitioning) or
     - drawback is asymmetry of partition as outer shell is usually narrow
  3. o is closer to p than to some other object $p_2 \in S$ (generalized hyperplane partitioning)
- Distances from pivots are useful in pruning the search



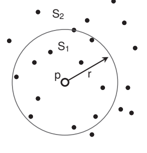


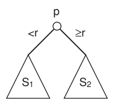
vp-tree (Metric tree; Uhlmann|Yianilos)

- Ball partitioning method
- Pick p from S and let r be median of distances of other objects from p
- Partition S into two sets S_1 and S_2 where:

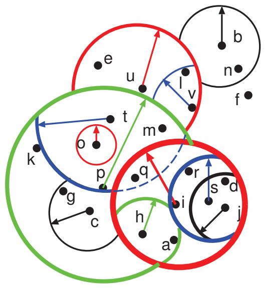
$$\begin{aligned} S_1 &= \{o \in S \setminus \{p\} \mid d(p, o) < r\} \\ S_2 &= \{o \in S \setminus \{p\} \mid d(p, o) \geq r\} \end{aligned}$$

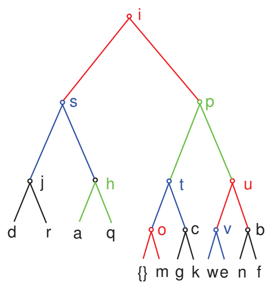
- Apply recursively, yielding a binary tree with pivot and radius values at internal nodes
- Choosing pivots
  1. simplest is to pick at random
  2. choose a random sample and then select median





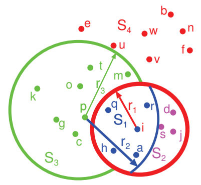
vp-tree Example

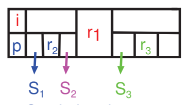




Increasing Fanout in vp-tree

- Fanout of a node in vp-tree is low
- Options
  1. increase fanout by splitting S into m equal-sized subsets based on $m + 1$ bounding values $r_0, \ldots, r_m$ or even let $r_0 = 0$ and $r_m = \infty$
  2. mvp-tree
     - each node is equivalent to collapsing nodes at several levels of vp-tree
     - use same pivot for each subtree at a level although the ball radius values differ
     - rationale: only need one distance computation per level to visit all nodes at the level (useful when search backtracks)
       a. first pivot i partitions into ball of radius r_1
       b. second pivot p partitions inside of the ball for i into subsets S_1 and S_2 , and outside of the ball for i into subsets S_3 and S_4





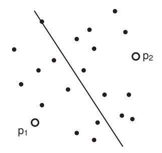
S_1 ={a,h,q,t}
S_2 ={d,j,s}
S_3 ={c,g,k,m,o,t}

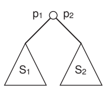
gh-tree (Metric tree; Uhlmann)

- Generalized hyperplane partitioning method
- Pick p_1 and p_2 from S and partition S into two sets S_1 and S_2 where:

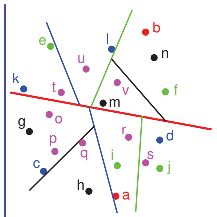
$$\begin{aligned} S_1 &= \{o \in S \setminus \{p_1, p_2\} \mid d(p_1, o) \leq d(p_2, o)\} \\ S_2 &= \{o \in S \setminus \{p_1, p_2\} \mid d(p_2, o) < d(p_1, o)\} \end{aligned}$$

- Objects in S_1 are closer to p_1 than to p_2 (or equidistant from both), and objects in S_2 are closer to p_2 than to p_1
  - hyperplane corresponds to all points o satisfying $d(p_1, o) = d(p_2, o)$
  - can also “move” hyperplane, by using $d(p_1, o) = d(p_2, o) + m$
- Apply recursively, yielding a binary tree with two pivots at internal nodes

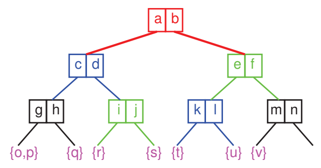




gh-tree Example



(a)



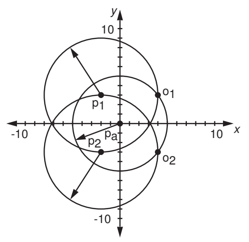
(b)

Increasing Fanout in gh-tree

- Fanout of a node in gh-tree is low
- Geometric Near-neighbor Access tree (GNAT; Brin)
  1. increase fanout by adding m pivots $P = \{p_1, \ldots, p_m\}$ to split S into $S_1, \ldots, S_m$ based on which of the objects in P is the closest
  2. for any object $o \in S \setminus P$, o is a member of S_i if $d(p_i, o) \le d(p_j, o)$ for all $j = 1, \ldots, m$
  3. store information about ranges of distances between pivots and objects in the subtrees to facilitate pruning search

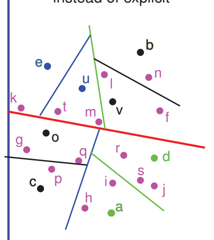
Bisector Tree (bs-tree) (Kalantari/McDonald)

1. gh-trees with covering balls
2. Drawback: covering ball of a node is sometimes larger than that of its ancestor (termed eccentric)
3. Bad for pruning as want radii of balls to decrease as search descends

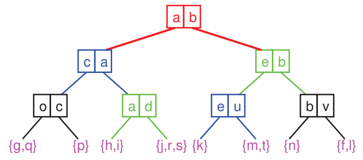


mb-tree (Dehne/Noltemeier)

1. Inherit one pivot from ancestor node
2. Fewer pivots and fewer distance computations but perhaps deeper tree
3. Like bucket (k) PR k-d tree as split whenever region has greater than k objects ($k > 1$) but region partitions are implicit (defined by pivot objects) instead of explicit



(a)

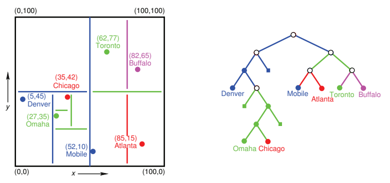


(b)

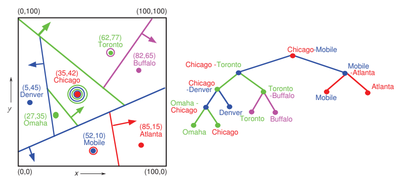
Comparison of mb-tree (BSP tree) and PR k-d tree

- Partition of underlying space analogous to that of BSP tree for points

PR k-d tree

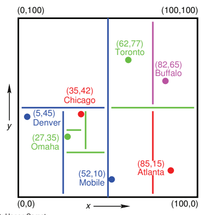


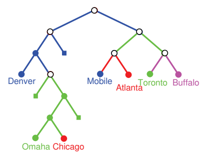
BSP tree



PR k-d tree

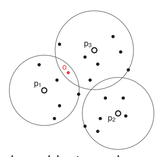
1. Regular decomposition point representation
2. Decompose whenever a block contains more than one point, while cycling through attributes
3. Maximum level of decomposition depends on minimum point separation
   - if two points are very close, then decomposition can be very deep
   - can be overcome by viewing blocks as buckets with capacity c and only decomposing a block when it contains more than c points





M-tree (Ciaccia et al.)

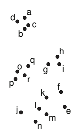
- Dynamic structure based on R-tree (actually SS-tree)
- All objects in leaf nodes
- Balls around "routing" objects (like pivots) play same role as minimum bounding boxes



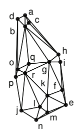
- Pivots play similar role as in GNAT, but:
  1. all objects are stored in the leaf nodes and an object may be referenced several times in the M-tree as it could be a routing object in more than one nonleaf node
  2. for an object o in a subtree of node n, the subtree's pivot p is not always the one closest to o among all pivots in n
  3. object o can be inserted into subtrees of several pivots: a choice
- Each nonleaf node n contains up to c entries of format (p, r, D, T)
  1. p is the pivot (i.e., routing object)
  2. r is the covering radius
  3. D is distance from p to its parent pivot p'
  4. T points to the subtree

Delaunay Graph

- Definition
  1. each object is a node and two nodes have an edge between them if their Voronoi cells have a common boundary
  2. explicit representation of neighbor relations that are implicitly represented in a Voronoi diagram
     - equivalent to an index or access structure for the Voronoi diagram
  3. search for a nearest neighbor of q starts with an arbitrary object and then proceeds to a neighboring object closer to q as long as this is possible
- Unfortunately we cannot construct Voronoi cells explicitly if only have interobject distances
- Spatial Approximation tree (sa-tree): approximation of the Delaunay graph



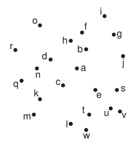
Point Set

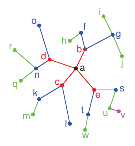


Deluanay graph

sa-tree (Navarro)

- Definition:
  1. choose arbitrary object a as root of tree
  2. find N(a), smallest possible set of neighbors of a, so that any neighbor is closer to a than to any other object in N(a)
     - i.e., x is in N(a) iff for all $y \in N(a) - \{x\}$, $d(x, a) < d(x, y)$
     - all objects in $S \setminus N(a)$ are closer to some object in N(a) than to a
  3. objects in N(a) become children of a
  4. associate remaining objects in S with closest child of a, and recursively define subtrees for each child of a



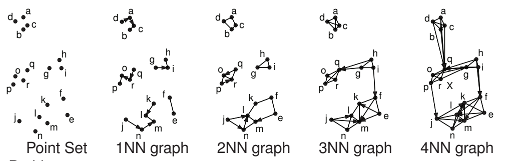


1. a is root
2. N(a)={b,c,d,e}
3. second level
4. h $\notin$ N(a) and N(b) as h closer to F than to b or a
5. fourth level

- Use heuristics to construct sa-tree as N(a) is used in the definition which makes it circular, and thus resulting tree is not necessarily minimal and not unique

kNN Graphs (Sebastian/Kimia)

1. Each vertex has an edge to each of its k nearest neighbors



Point Set 1NN graph 2NN graph 3NN graph 4NN graph

2. Problems
   - graph is not necessarily connected
   - even if increase k so graph is connected, search may halt at object p which is closer to q than any of the k nearest neighbors of p but not closer than all of the objects in p's neighbor set (e.g., the $k+1^{st}$ nearest neighbor)
     - Ex: search for nearest neighbor of X in 4NN graph starting at any one of {e,f,j,k,l,m,n} will return k instead of r
   - overcome by extending size of search neighborhood as in approximate nearest neighbor search
   - use several starting points for search (i.e., seeds)
3. Does not require triangle inequality and thus works for arbitrary distances

Alternative Approximations of the Delaunay Graph

1. Other approximation graphs of the Delaunay graph are connected by virtue of being supersets of the minimal spanning tree (MST) of the vertices
2. Relative neighborhood graph (RNG): an edge between vertices u and v if for all vertices p, u is closer to v than is p or v is closer to u than is p — that is, $d(u,v) \leq Max\{d(p,u), d(p,v)\}$
3. Gabriel graph (GG): an edge between vertices u and v if for all other vertices p we have that $d(u,p)^2 + d(v,p)^2 \geq d(u,v)^2$
4. RNG and GG are not restricted to Euclidean plane or Minkowski metrics
5. MST(E) ⊂RNG(E) ⊂GG(E) ⊂DT(E) in Euclidean plane with edges E
6. MST(E) ⊂RNG(E) ⊂GG(E) in any metric space as DT is only defined for the two-dimensional Euclidean plane

Use of Delaunay Graph Approximations

1. Unless approximation graph is a superset of Delaunay graph (which it is not), to be useful in nearest neighbor searching, we need to be able to force the algorithm to move to other neighbors of current object p even if they are farther from q than p
2. Examples:
   - kNN graph: use extended neighborhood
   - sa-tree: prune search when can show (with aid of triangle inequality) that it is impossible to reach the nearest neighbor via a transition to nearest neighbor or set of neighbors
   - RNG and GG have advantage that are always connected and don't need seeds
   - advantage of kNN graph is that k nearest neighbors are precomputed

Spatial Approximation Sample Hierarchy (SASH)(Houle)

- Hierarchy of random samples of set of objects S of size $S/2, S/4, S/8, \ldots, 1$
- Makes use of approximate nearest neighbors
- Has similar properties as the kNN graph
  1. both do not require that the triangle inequality be satisfied
  2. both are indexes
     - $O(N^2)$ time to build kNN graph as no existing index
     - SASH is built incrementally level by level starting at root with samples of increasing size making use of index already built for existing levels thereby taking $O(N \log_2 N)$ time
     - each level of SASH is a kNN tree with maximum $k = c$
- Key to approximation is to treat the "nearest neighbor relation" as an "equivalence relation" even though this is not generally true
  1. assumption of "equivalence" relation is the analog of ϵ
  2. no symmetry: x being approximate nearest neighbor of x' does not mean that x' must be an approximate nearest neighbor of x
  3. no transitivity: x being approximate nearest neighbor of q and x' being approximate nearest neighbor of x does not mean that x' must be an approximate nearest neighbor of q
  4. construction of SASH is analog of UNION operation
  5. finding approximate nearest neighbor is analog of FIND operation

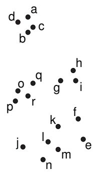
SASH vis-a-vis Triangle Inequality

- Triangle inequality is analogous to transitivity with $\leq$ corresponding to "approximate nearest neighbor" relation
- Appeal to triangle inequality, $d(x',q) \leq d(q,x) + d(x',x)$, regardless of whether or not it holds
  1. to establish links to objects likely to be neighbors of query object q,
     - when $d(q,x)$ and $d(x',x)$ are both very small, then $d(q,x')$ is also very small (analogous to "nearest")
     - implies if $x \in S \setminus S'$ is a highly ranked neighbor of both q and $x' \in S'$ among objects in $S \setminus S'$, then x' is also likely to be a highly ranked neighbor of q among objects in S'
       - x' is a highly ranked neighbor of x (symmetry)
       - AND x is a highly ranked neighbor of q
       - RESULT: x' is a highly ranked neighbor of q (transitivity)
  2. INSTEAD of to eliminate objects that are guaranteed not to be neighbors

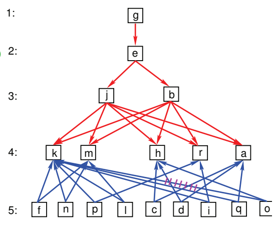
Mechanics of SASH

- SASH construction (UNION of UNION-FIND)
  1. form hierarchy of samples
  2. assume SASH$_i$ has been built and process sample S'
     - know that x in SASH$_i$\SASH$_{i-1}$ is one of p approximate nearest neighbors of $x' \in S'$ and use SASH$_i$ to determine x
     - infer that x' is one of $c > p$ approximate nearest neighbors in S' of x (symmetry)
  3. special handling to ensure that every object at level $i+1$ is an approximate nearest neighbor of at least one object at level i (i.e., no orphan objects)
- Finding k approximate nearest neighbors of q (FIND of UNION-FIND)
  1. follow links from level $i-1$ of SASH to level i retaining in U_i the k_i approximate nearest neighbors of q at level i of the SASH
  2. determine k approximate nearest neighbors of q from the union of U_i over all levels of the SASH
  3. know that x in U_i is an approximate nearest neighbor of q
  4. know that x' in U_{i+1} is an approximate nearest neighbor of x in U_i
  5. infer that x' in U_{i+1} is an approximate nearest neighbor of q (transitivity)

Example of SASH construction



- Ex: P=2 C=5
- Initially, no choice in the first 3 levels
- Find two closest objects at level 4 for each object at level 5
  - f:k,m n:k,m
  - p:k,r l:k,m
  - c:a,h d:a,h
  - i:h,k d:h,r
  - o:k,r
- Retain 5 nearest neighbors at level 5 to each object at level 4
  - k:{f,n,p,l,i,~~o~~ }
  - m:f,n,l
  - n:c,d,i,q
  - a:c,d
- Ignore o as k has 5 closer neighbors

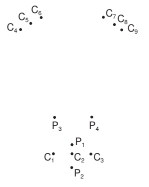


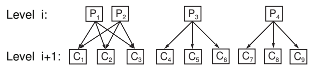
Example SASH Approximate k Nearest Neighbor Finding

- Ex: $k = 3$ and query object c
- Let $f(k,i) = k_i = k^{1-(h-i)/\log_2 N}$ yielding $k_i = (1,1,2,2,3)$
- $U_1 =$ root g of SASH
- $U_2 =$ objects reachable from U_1 which is e
- $U_3 =$ objects reachable form U_2 which is b and j which are retained as $k_3 = 2$
- U_4 objects reachable from U_3 which is {a,h,k,m,r} and we retain just a and h in U_4 as $k_4 = 2$
- $U_5 =$ objects reachable form U_4 which is {c,d,i,q}, and we retain just c, d, and q in U_5 as $k_5 = 3$
- Take union of U_1, U_2, U_3, U_4, U_5 which is the set {a,b,c,d,e,g,h,i,j,k,m,q,r}, and the closest three neighbors to query object c are a, b, and d

Drawback of SASH

- Assumes that if a at level i is an approximate nearest neighbor of o at level $i+1$, then by symmetry o is likely to be an approximate nearest neighbor of a, which is not generally true
- Ex: objects at level i are not necessarily linked to their nearest neighbors at level $i+1$





- P$_3$ and P$_4$ at level i are linked to the sets of three objects $\{C_4, C_5, C_6\}$ and $\{C_7, C_8, C_9\}$, respectively, at level i+1, instead of to their nearest neighbors C$_1$, C$_2$, and C$_3$ at level i+1.

Part D: Nearest Neighbor Finding

1. Classical methods such as branch and bound
2. K nearest neighbors
3. Incremental nearest neighbor finding
   - General method
   - Permitting duplicate instances of objects
4. Approximate nearest neighbor finding
5. Probably approximately correct (PAC) nearest neighbor finding

Branch and Bound Algorithm (Fukunaga/Narendra)

1. Visit elements in hierarchy using a depth-first traversal
   - maintain a list L of current candidate k nearest neighbors
2. D_k: distance between q and the farthest object in L
   - $D_k = \max_{o \in L}\{d(q,o)\}$, or ∞ if L contains fewer than k objects
   - D_k is monotonically non-increasing over the course of the search traversal, and eventually reaches the distance of the k^{th} nearest neighbor of q
3. If element e_t being visited represents an object o (i.e., $t = 0$), then insert o into L, removing farthest if $|L| > k$
4. Otherwise, e_t ($t \geq 1$) is not an object
   - construct an *active list* $A(e_t)$ of child elements of e_t, ordered by "distance" from q
   - recursively visit the elements in $A(e_t)$ in order, backtracking when
     a. all elements have been visited, or
     b. reaching an element $e_{t'} \in A(e_t)$ with $d_{t'}(q, e_{t'}) > D_k$
        - condition ensures that all objects at distance of k^{th} nearest neighbor are reported
        - if sufficient to report k objects, then use $d_{t'}(q, e_{t'}) \geq D_k$

Incremental Nearest Neighbors (Hjaltason/Samet)

- Motivation
  1. often don't know in advance how many neighbors will need
  2. e.g., want nearest city to Chicago with population > 1 million
- Several approaches
  1. guess some area range around Chicago and check populations of cities in range
     - if find a city with population > 1 million, must make sure that there are no other cities that are closer with population > 1 million
     - inefficient as have to guess size of area to search
     - problem with guessing is we may choose too small a region or too large a region
       a. if size too small, area may not contain any cities with right population and need to expand the search region
       b. if size too large, may be examining many cities needlessly
  2. sort all the cities by distance from Chicago
     - impractical as we need to re-sort them each time pose a similar query with respect to another city
     - also sorting is overkill when only need first few neighbors
  3. find k closest neighbors and check population condition

Mechanics of Incremental Nearest Neighbor Algorithm

- Make use of a search hierarchy (e.g., tree) where
  1. objects at lowest level
  2. object approximations are at next level (e.g., bounding boxes in an R-tree)
  3. nonleaf nodes in a tree-based index
- Traverse search hierarchy in a "best-first" manner similar to A*-algorithm instead of more traditional depth-first or breadth-first manners
  1. at each step, visit element with smallest distance from query object among all unvisited elements in the search hierarchy
     - i.e., all unvisited elements whose parents have been visited
  2. use a global list of elements, organized by their distance from query object
     - use a priority queue as it supports necessary insert and delete minimum operations
     - ties in distance: priority to lower type numbers
     - if still tied, priority to elements deeper in search hierarchy

Incremental Nearest Neighbor Algorithm

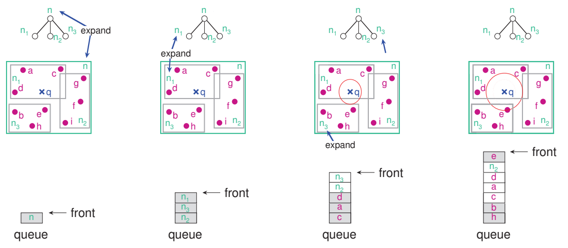
Algorithm:

```
INCNEAREST($q$, $S$, $T$)
1  $Q \leftarrow$ NEWPRIORITYQUEUE()
2  $e_t \leftarrow$ root of the search hierarchy induced by $q$, $S$, and $T$
3  ENQUEUE($Q$, $e_t$, 0)
4  while not ISEMPTY($Q$) do
5    $e_t \leftarrow$ DEQUEUE($Q$)
6    if $t = 0$ then /* $e_t$ is an object */
7      Report $e_t$ as the next nearest object
8    else
9      for each child element $e_{t'}$ of $e_t$ do
10       ENQUEUE($Q$, $e_{t'}$, $d_{t'}(q, e_{t'})$)
```

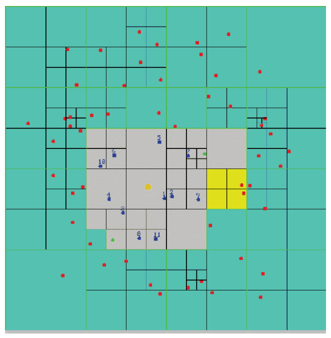
1. Lines 1-3 initialize priority queue with root
2. In main loop take element e_t closest to q off the queue
   - report e_t as next nearest object if e_t is an object
   - otherwise, insert child elements of e_t into priority queue

Example of INCNEAREST

- Initially, algorithm descends tree to leaf node containing q
  - expand n
  - expand n$_1$
- Start growing search region
  - expand n$_3$
  - report e as nearest neighbor



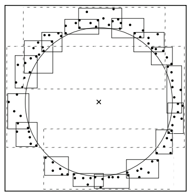
VASCO Spatial Applet



http://www.cs.umd.edu/users/hjs/quadtree/index.html

Complexity Analysis

- Algorithm is I/O optimal
  - no nodes outside search region are accessed
  - better pruning than branch and bound algorithm
- Observations for finding k nearest neighbors for uniformly-distributed two-dimensional points
  - expected # of points on priority queue: $O(\sqrt{k})$
  - expected # of leaf nodes intersecting search region: $O(k + \sqrt{k})$
- In worst case, priority queue will be as large as entire data set
  - e.g., when data objects are all nearly equidistant from query object
  - probability of worst case very low, as it depends on a particular configuration of both the data objects and the query object (but: curse of dimensionality!)



Duplicate Instances of Objects

- Objects with extent such as lines, rectangles, regions, etc. are indexed by methods that associate the objects with the different blocks that they occupy
- Indexes employ a disjoint decomposition of space in contrast to non-disjoint as is the case for bounding box hierarchies (e.g., R-tree)
- Search hierarchies will contain multiple references to some objects
- Adapting incremental nearest neighbor algorithm:
  1. make sure to detect all duplicate instances that are currently in priority queue
  2. avoid inserting duplicate instances of an object that has already been reported

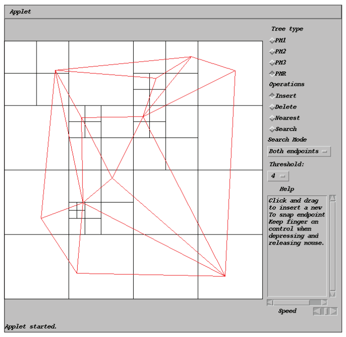
Duplicate Instances Algorithm

```
INCNEARESTDUP($q$, $S$, $T$)
1  $Q \leftarrow$ NEWPRIORITYQUEUE()
2  $e_t \leftarrow$ root of the search hierarchy induced by $q$, $S$, and $T$
3  ENQUEUE($Q$, $e_t$, 0)
4  while not ISEMPTY($Q$) do
5    $e_t \leftarrow$ DEQUEUE($Q$)
6    if $t = 0$ then /* $e_t$ is an object */
7      while $e_t$ = FIRST($Q$) do
8        DELETEFIRST($Q$)
9      Report $e_t$ as the next nearest object
10   else /* $e_t$ is not an object */
11     for each child element $e_{t'}$ of $e_t$ do
12       if $t' > 0$ or $d_{t'}(q, e_{t'}) \geq d_t(q, e_t)$ then
13         ENQUEUE($Q$, $e_{t'}$, $d_{t'}(q, e_{t'})$)
```

Differences from INCNEAREST

1. Object o ($e_{t'}$) is enqueued only if o has not yet been reported
   - check if o's distance from q is less than distance from e_t to q (line 12)
   - if yes, then o must have been encountered in an element $e_{t''}$ which was closer to q and hence already been reported
2. Check for multiple instances of object o and report only once (lines 7–9)
3. Order objects in queue by identity when at same distance
4. Retrieve all nodes in the queue before objects at same distance
   - important because an object can have several ancestor nodes of the same type
   - interesting as unlike INCNEAREST where want to report neighbors as soon as possible so break ties by giving priority to elements with lower type numbers

VASCO Spatial Applet



Click and drag to insert a new To snap endpoint Keep finger on control when depressing and releasing mouse.

http://www.cs.umd.edu/users/hjs/quadtree/index.html

INCNEAREST Extensions

1. Incremental range query
2. Incremental retrieval of k nearest neighbors
   - need an extra queue to keep track of k neighbors found so far and can use distance d_k from q of the k^{th} candidate nearest neighbor o_k to reduce number of priority queue operations
3. Farthest neighbor
4. Pairs of objects
   - distance join
   - distance semi-join

Approximate Nearest Neighbors

1. Often, obtaining exact results is not critical and willing to trade off accuracy for improved performance
2. Let ϵ denote the approximation error tolerance
   - common criterion is that the distance between q and the resulting candidate nearest neighbor o' is within a factor of $1 + \epsilon$ of the distance to the actual nearest neighbor o
   - i.e., $d(q, o') \leq (1 + \epsilon)d(q, o)$

Approximate Nearest Neighbors with INCNEAREST

1. Modify INCNEAREST by multiplying the key values for non-object elements on the priority queue by $1 + \epsilon$
   - in a practical sense, non-object element e_t is enqueued with a larger distance value — that is, by a factor of $(1 + \epsilon)$
   - implies that we delay its processing, thereby allowing objects to be reported 'before their time'
   - e.g., once e_t is finally processed, all objects o satisfying $d(q, o) \leq (1 + \epsilon)d_t(q, e_t)$ (which is greater than $d_t(q, e_t)$ if $\epsilon > 0$) would have already been reported
   - thus an object c in e_t with a distance $d(q, c) \leq d(q, o)$ could exist, yet o is reported before c
   - algorithm does not necessarily report the resulting objects in strictly increasing order of their distance from q
2. Different from Arya/Mount algorithm which cannot be incremental as priority queue only contains non-object elements
   - shrinks distance r from q to the closest object o by a factor of $1 + \epsilon$ and only inserts a non-object element e into the priority queue if the distance $d(b, q)$ of e's corresponding block b from q is less than the shrunken distance

Probably Approximately Correct (PAC) Nearest Neighbors (Ciaccia/Patella)

- Relax approximate nearest neighbor condition by stipulating a maximum probability δ for tolerating failure, thereby enabling the decision process to halt sooner at the risk δ of being wrong
- Object o' is considered a PAC-nearest neighbor of q if the probability that $d(q,o') \leq (1+\epsilon)\cdot d(q,o)$ is at least $1-\delta$, where o is actual nearest neighbor
- Alternatively, given ϵ and δ, $1-\delta$ is the minimum probability that o' is the $(1+\epsilon)$-approximate nearest neighbor of q
- Ciaccia and Patella use information about the distances between q and the data objects to derive an upper bound s on the distance between q and a PAC-nearest neighbor o'
- Distance bound s is used during the actual nearest neighbor search as a pre-established halting condition — that is, the search can be halted once locating an object o' with $d(q,o') \leq s$
- Method is analogous to executing a variant of a range query, where the range is defined by the distance bound s, which halts on the first object in the range
- Difficulty is determining a relationship between δ and the distance bound s

Concluding Remarks

1. Similarity search is a broad area of research
2. Much relation to geometry; geometric setting is usually missing
3. Progress is heavily influenced by applications
4. Need to look at old literature to be able to evaluate current research results
5. Much is left to do as difficult to say what is best solution

Selected Overview References

- H. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS.* Addison-Wesley, Reading, MA, 1990.
- H. Samet. *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, MA, 1990.
- V. Gaede and O. Günther. Multidimensional access methods. *ACM Computer Surveys*, 20(2):170–231, June 1998.
- C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322–373, Sept. 2001.
- E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–322, Sept. 2001.
- G. R. Hjaltason and H. Samet. Properties of embedding methods for similarity searching in metric spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):530–549, May 2003. Also University of Maryland Computer Science TR-4102.
- G. R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. *ACM Transactions on Database Systems*, 28(4):517–580, Dec. 2003.
- H. Samet. *Foundations of Multidimensional and Metric Data Structures*, Morgan-Kaufmann, San Francisco, CA, 2006 (see overleaf).