Approximate Nearest Neighbors

Sariel Har Peled: Notes

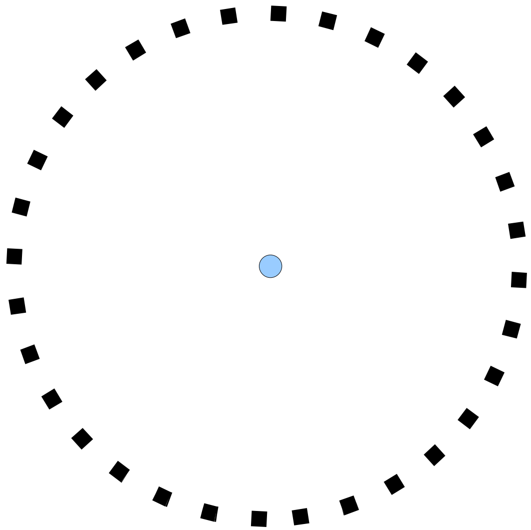
Arya, Mount, Netenyahu, Silverman, Wu *An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions*

Approximate Nearest Neighbors

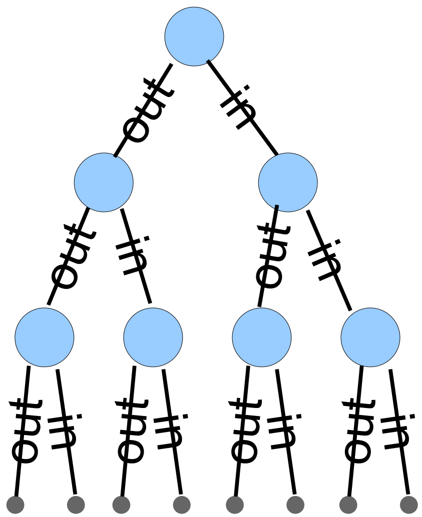
- What we want
  - O(n log n) preprocess
  - O(n) space
  - O(log n) time query
- Possible in 1 and 2D
- Not really in 3D

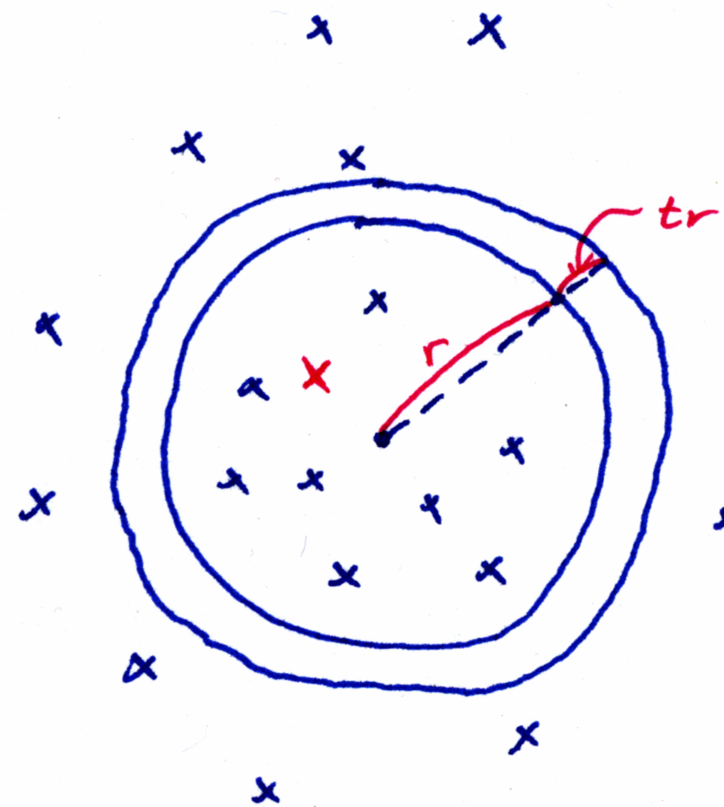
Lets Approximate

- Return a point within distance $(1+\varepsilon)r$
- Can achieve the bounds several ways
- First
  - compute rough approximation
  - use it to set scale for final solution
- Second
  - build a tree which solves the problem



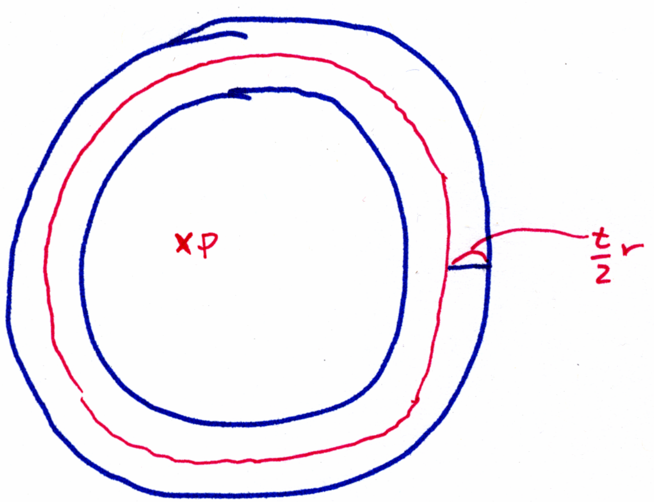
Ring Separator Tree





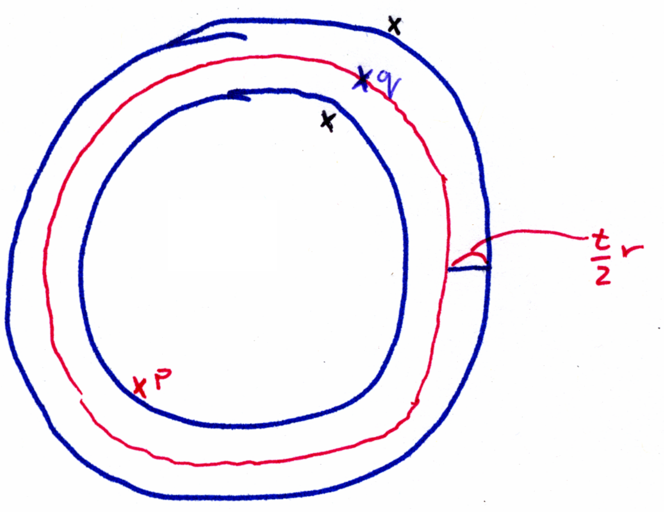
Ring Separator Tree

- Answer (1+4/t)-ANN queries in O(height)
- Check if rep is closest, if so update closest
- Recurse on correct side of halfway ball



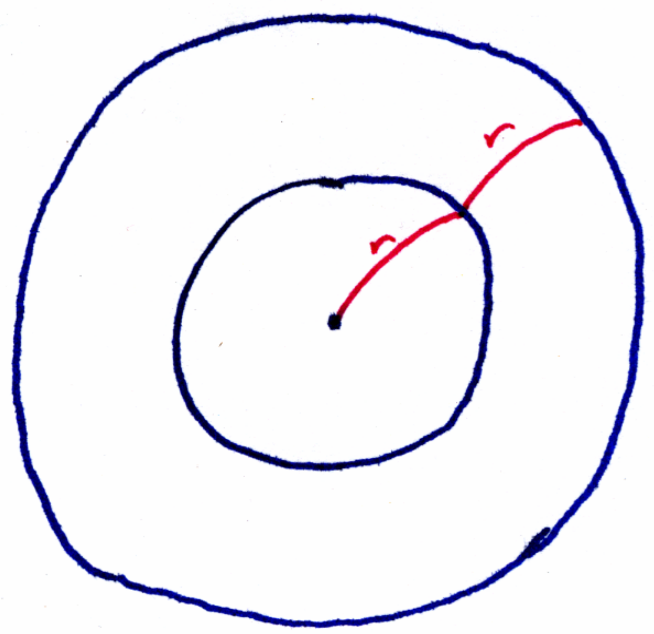
Error Bounds

- Closest: rt/2
- Returned: 2r+rt/2



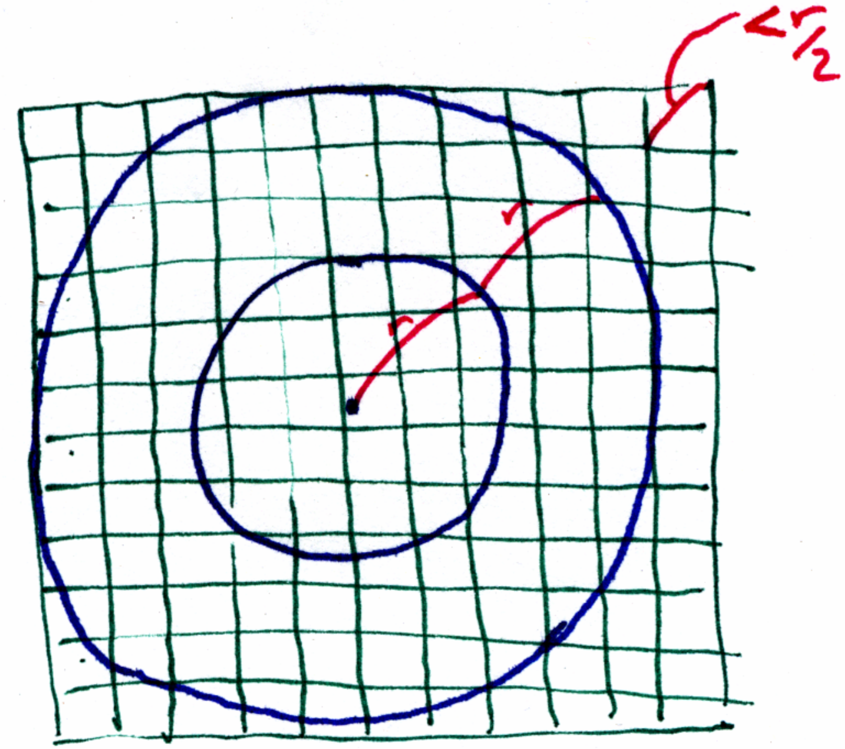
Construction

- Find circle contair n/c points



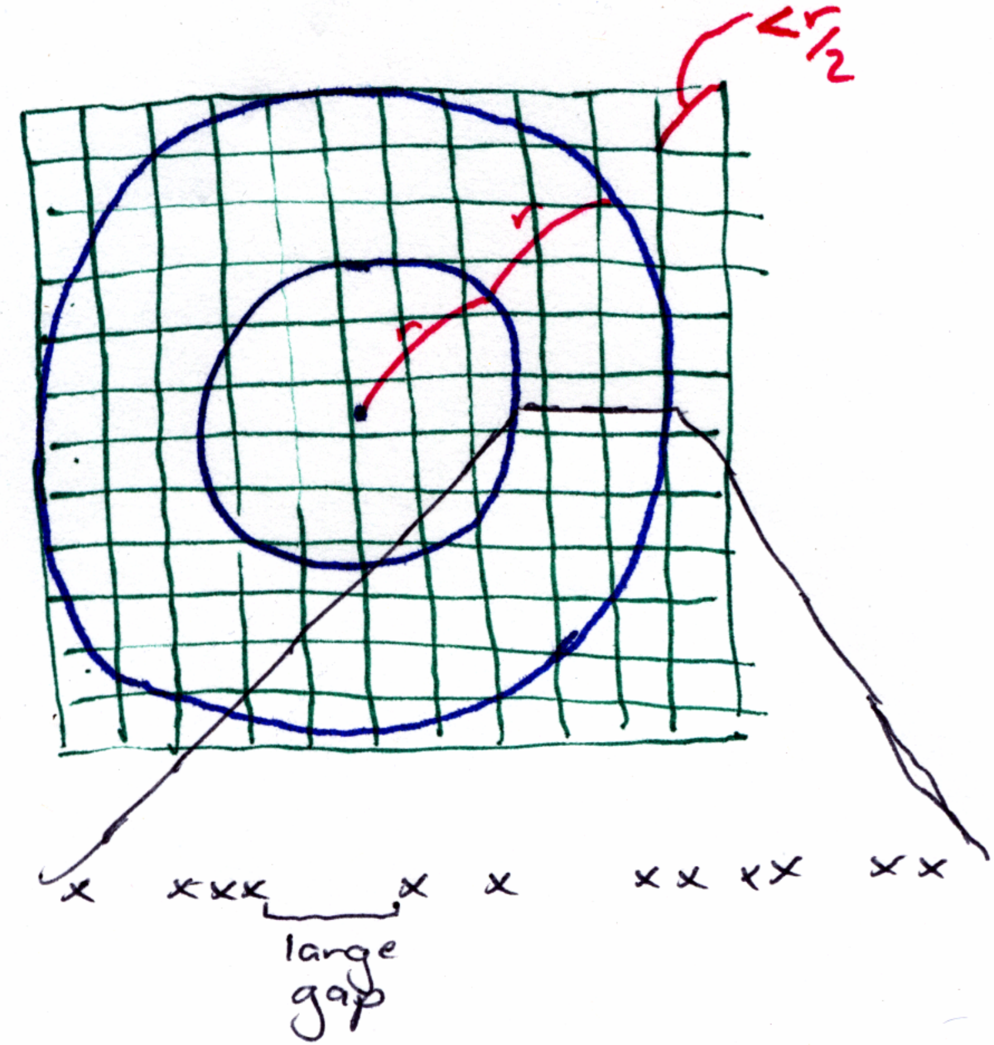
Construction

- Grid of side $L = \dfrac{r}{16\sqrt{d}}$
- Number of points $$\frac{(4L)^d n}{c}$$
- Set $c = 2(4\mathrm{L})^d$
- Ring has n/2 points



Construction

- Put ring in largest gap
- Size 2r/n



The Upshot

- Can preprocess in O(n log n) time
- Query time is O(log n)
- (4n+1) approximation!
- Amazingly, this is good enough

Bounded Distance

- Normal quadtree gives $O(\frac{1}{\epsilon^d} + \log\delta)$
- Why?
  - Approximation and r eliminates small cells (ε/4)r
  - Bound number of cells visited by last level
  - Do some algebra to get bound…

A Complete Algorithm

- Build
  - a compressed quadtree/finger tree
  - a ring separator tree
- Compute approximate value, R
- Start from
  - nodes of size approximately R
  - and closer than R to query point

Arya and Mount

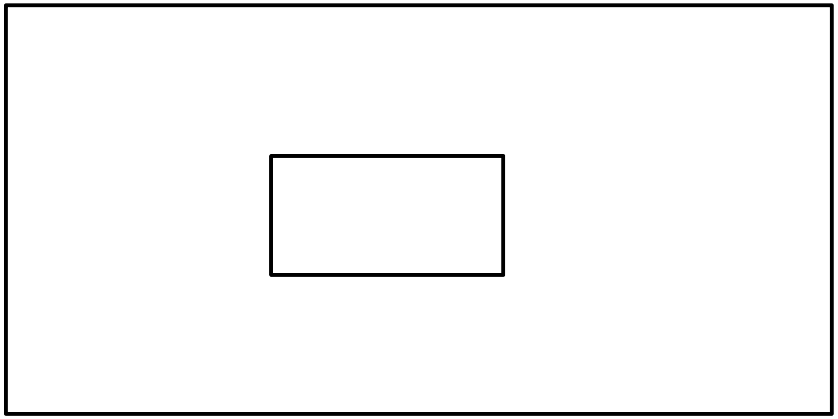
- $O(dn \log n)$ time
- $O(dn)$ space
- $O(c_{d,\varepsilon} \log n)$ time ANN
  - where $c_{d,\varepsilon} \leq d(1+6d/\varepsilon)^d$
- Can find k NN
- Any Minkowski metric
- Preprocessing does not depend on ε or metric

Overview

- Build BBD tree
- Locate leaf containing q
- Try nearby nodes in order of distance
- Stop when no node is close enough

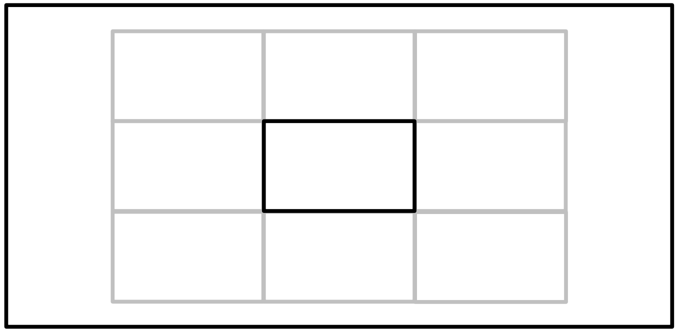
Tree types

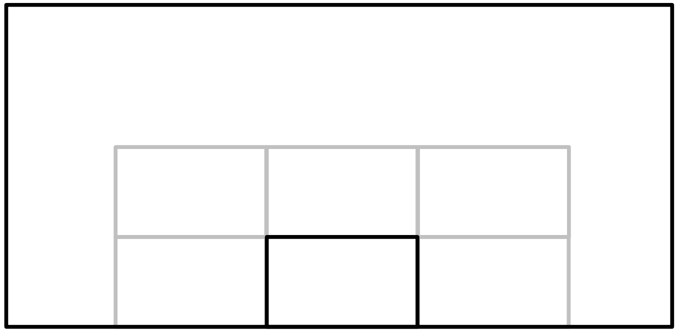
- KD reduce number of points each level
- Quadtree reduces size
- BBD does both
  - either KD-like split
  - or shrink



Properties

- Bounded aspect ratio
  - bound number of cells intersecting a volume
- Stickiness
  - control number of nearby cells
- Inner boxes not cut by children
  - so everything packs



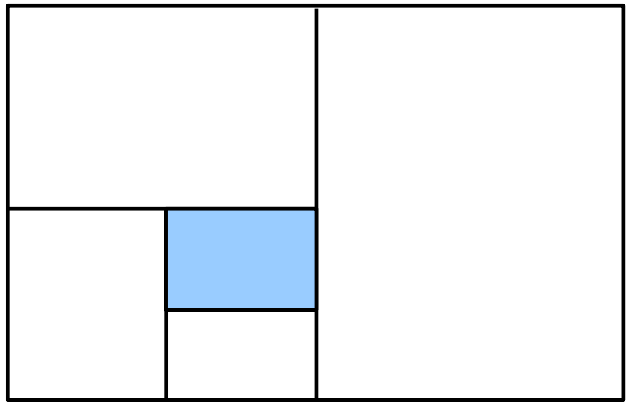


An Important Trick

- Maintain 3 sorted lists of points (x,y,z)
- Have links between lists
- Allows
  - removal of first k points in time k
  - O(d) time determination of min bounding box

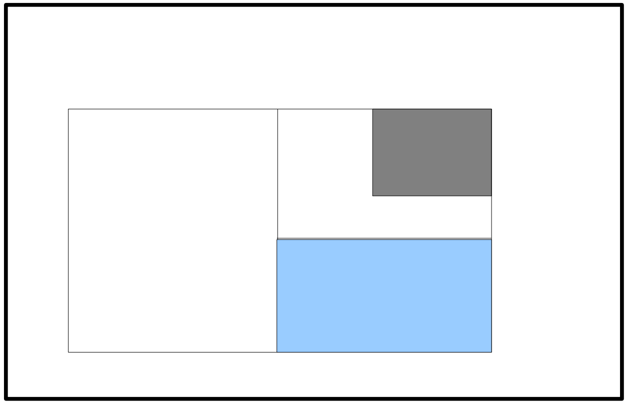
Computing Shrinks

- Compute a set of splits
  - until have n/c in a rectangle
  - trivially sticky



- Problems
  - doesn't respect nesting
  - may have to split many times

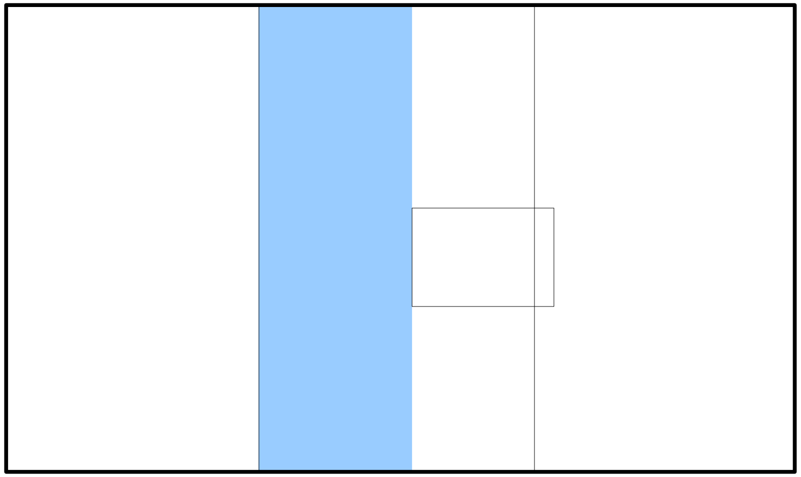
Computing Shrinks II



- Alway cut min enclosing box
  - constant time
  - always remove points
  - make sure it respects stickyness
- Include parent inner rectangle
  - go until it is cut out

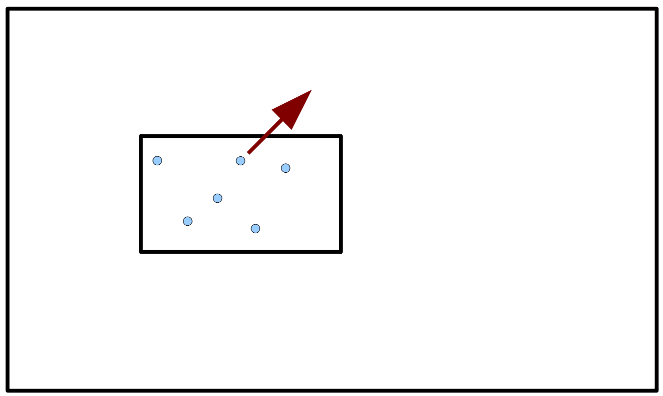
Computing Shrinks 2

- More flexible
- Shrink roughly as before



Tweaks

- Collapse trivial splits/shrinks
  - now no sequence of trivial splits
- Assign one point to each leaf
  - even to empty shrink cells

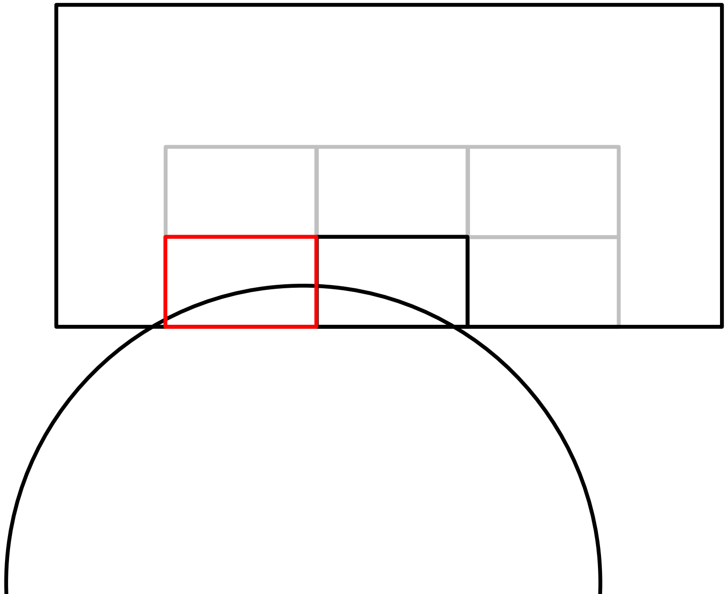


Properties

- Bounded occupancy
- Point near each leaf
- Can do point location in $O(d \log n)$ time
- Packing constraint
- Distance enumeration

Proof of Packing

- Ball of radius r
  - intersects $(1+6r/s)^d$ leaves of size s
- Trivial packing argument except for shrinks
  - use stickiness to replace outer boxes



ANN using BBD

- Number of leaves visited is $O((1+6d/\varepsilon)^d)$
- r is distance to last non-terminating leaf
- $r(1+\varepsilon) \leq dist(q,p)$
- Can't have visited cell smaller than $r\varepsilon/d$
  - this cell must have a point closer than $r(1+\varepsilon)$
- Use packing argument from before

Experimental Results

- Choices
  - shrink only when necessary
  - leaves held 5-8 points
- Results
  - Slightly slower than Kd trees for even data
  - Much faster for clustered data (10x or so)
  - Slightly slower than Kd trees for surfaces (20%)