Outline

- Introduction
- Exact Query Processing
- Approximate Query Processing
- Selectivity Estimation
- Open Problems

Approximate Query Processing

- Space Partitioning-based
  - Tree
  - Encoding
  - Locality Sensitive Hashing
- Graph-based Methods

Notes:

- Recent works mainly in the Database area
- Prefer ease of exposition over rigor
- Categorization is not fixed/unique

Space Partitioning-based

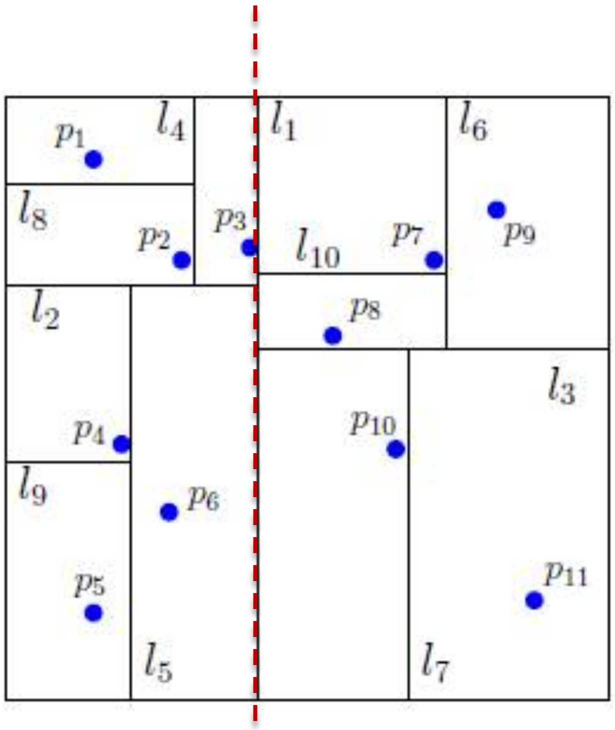
- Partition the whole space into partitions that cover the whole space
- Further divided into 3 sub-categories:
  - Tree-based
  - Encoding-based
  - Locality sensitive hashing-based

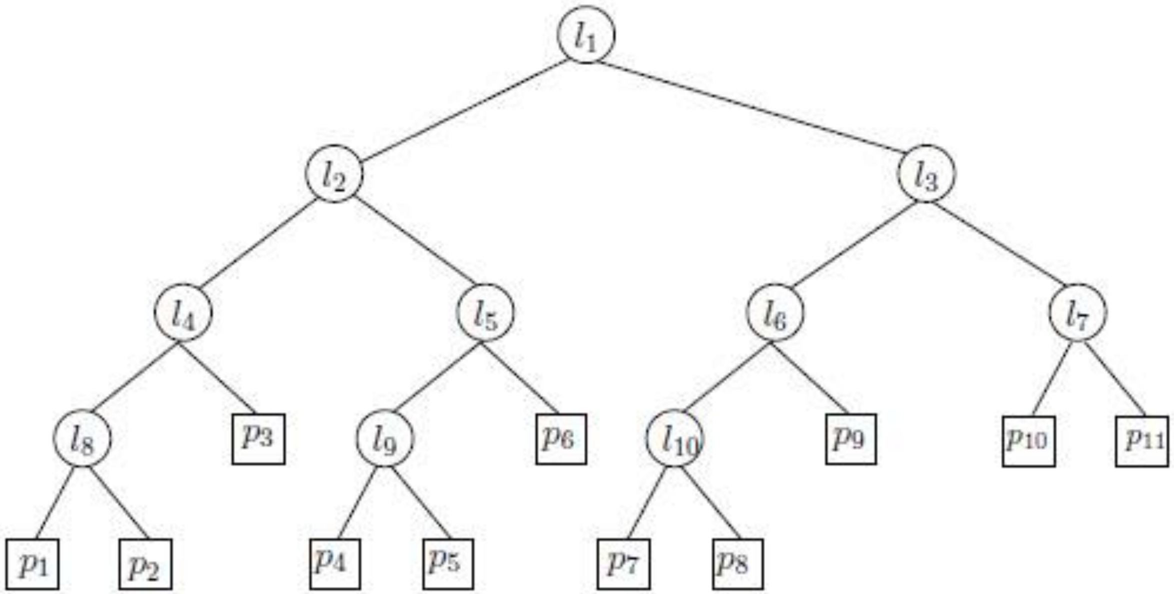
Tree-based

- Hierarchically partition the whole space into partitions that covers the whole space
- A natural idea in low-dimensional space
  - disjoint: kd-tree → Randomized kd-trees and variants
  - overlapping: R-tree → M-tree, Cover Tree, Spill tree

Problem:
Non-trivial modification needed to handle high-dimensional data

kd-tree Examples (low dimensional space)





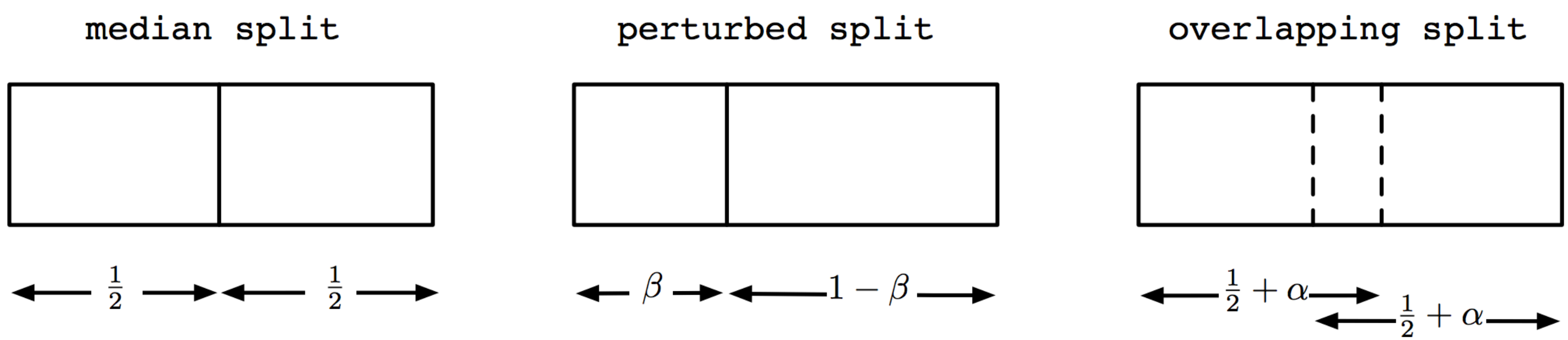
Step 1

- Mapping
  - Random top-k dimensions: Randomized kd-tree
  - PCA: PCA-tree
  - Random Rotation: NKD-Tree
  - Optimized Sparse Rotation: TP-Tree
  - Random Projection: RP-Tree

Main idea:
maximize the variance before the split

Step 2

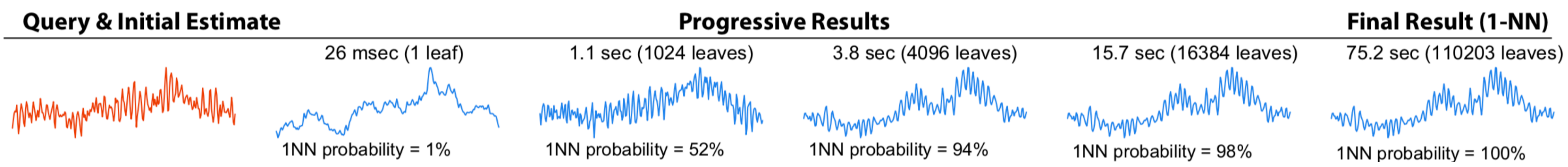
- Split
  - Dim 1
    - Median split: (randomized) KD-tree, PCA-tree, …
    - Perturbed split: RP-tree
    - Overlapping split: Spill Tree [DS15]
      - Virtual spill tree: “Spill” at query time
  - Dim 2:
    - Linear split
    - Non-linear split: [DIRW20]



Steps 3 & 4

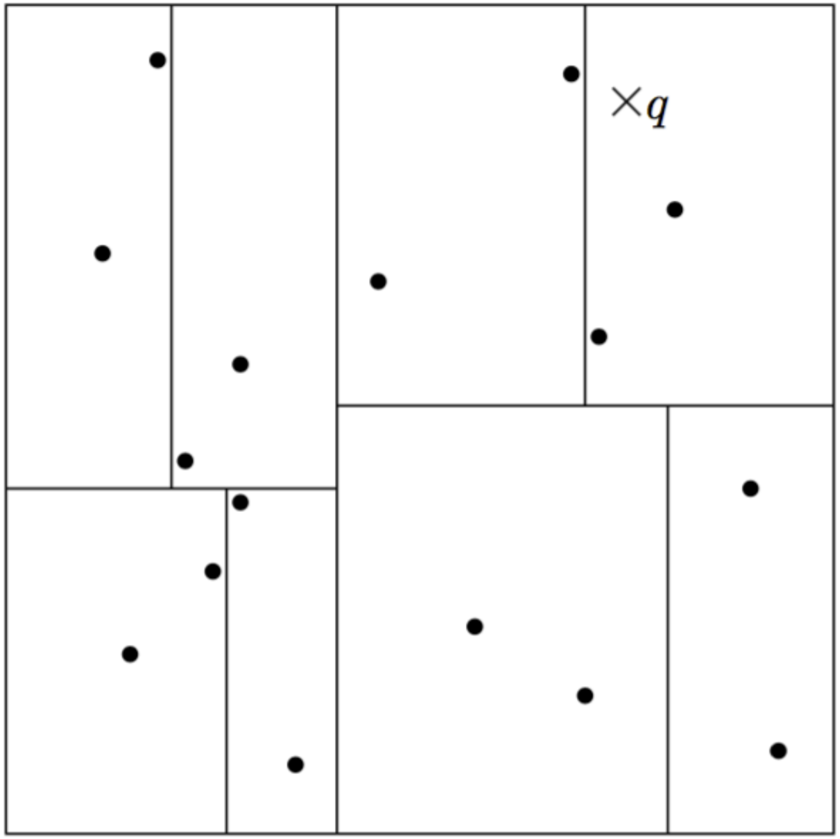
- (Optional) Tree ➔ Forest
  - Can be applied to all kinds of trees
  - Can use best-first search to coordinate the searches
- When to stop?
  - Guaranteed NN found
  - Bounded cost
  - Judged by a prediction model [LZAH20, GTEB+20]

GTEB+20

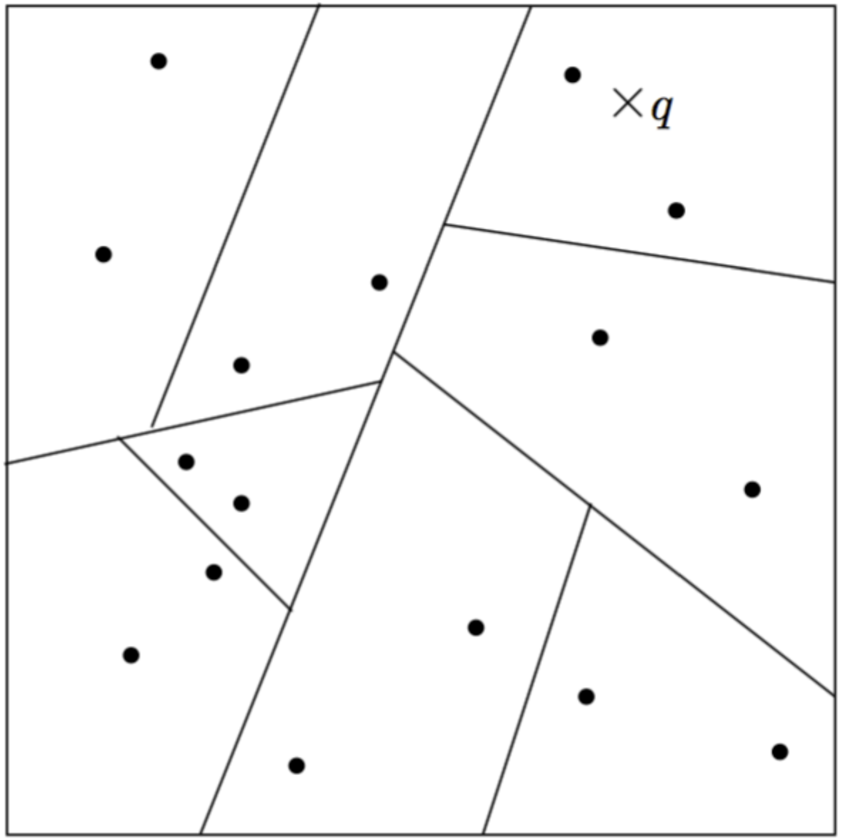


RP-tree Example

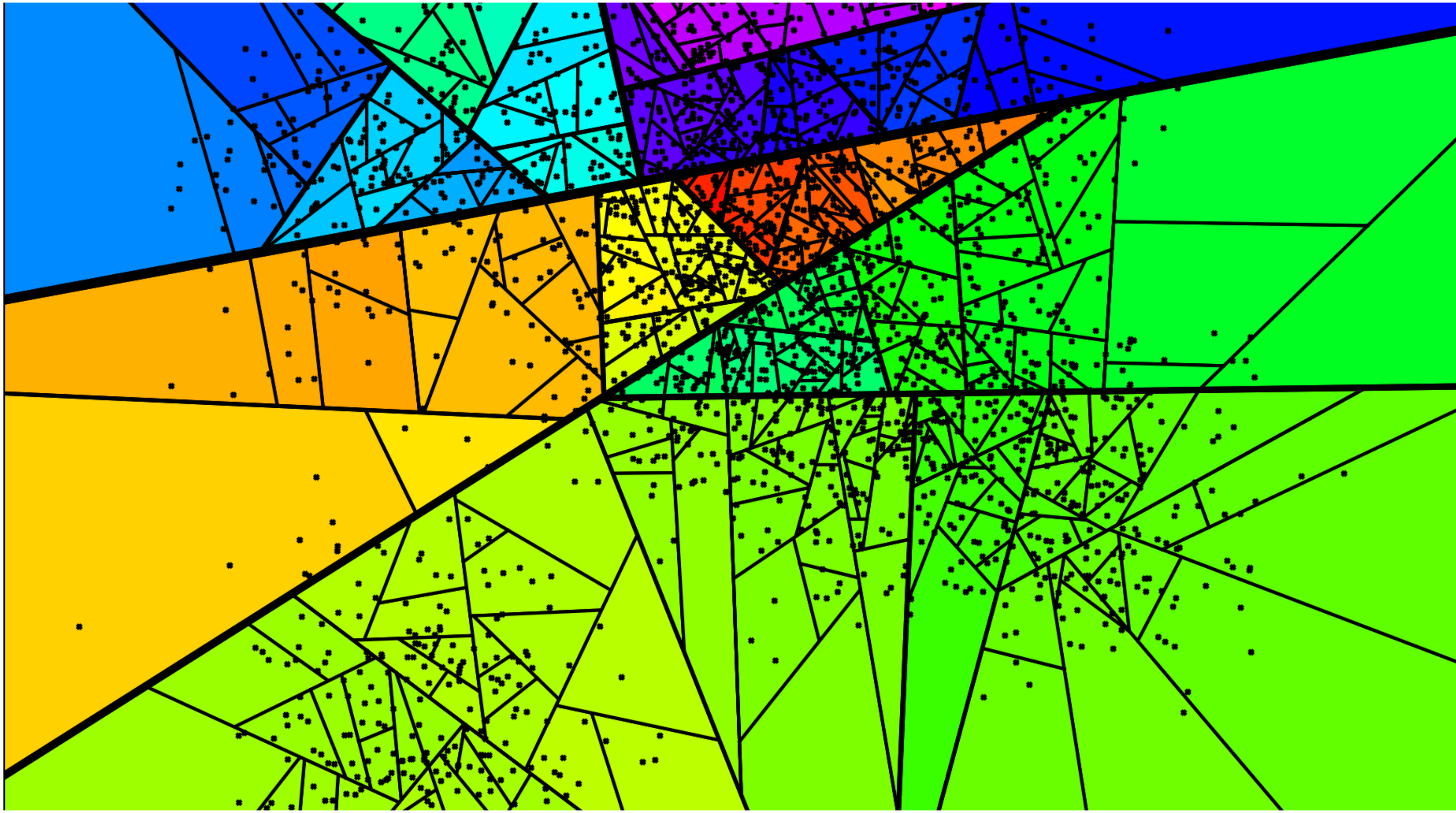
kd-tree



rp-tree



Annoy Example



Erik Bernhardsson, “Approximate nearest neighbor methods and vector models”, 2015

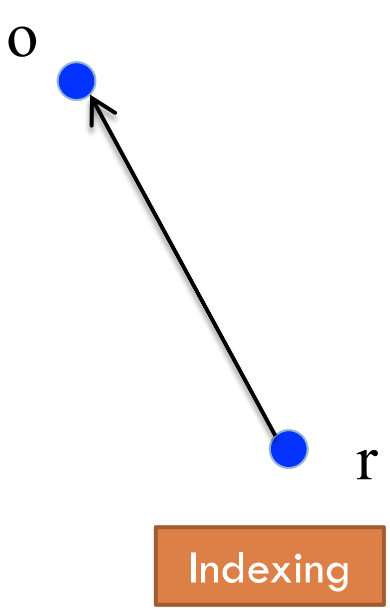
Trees with Overlapping Partitions

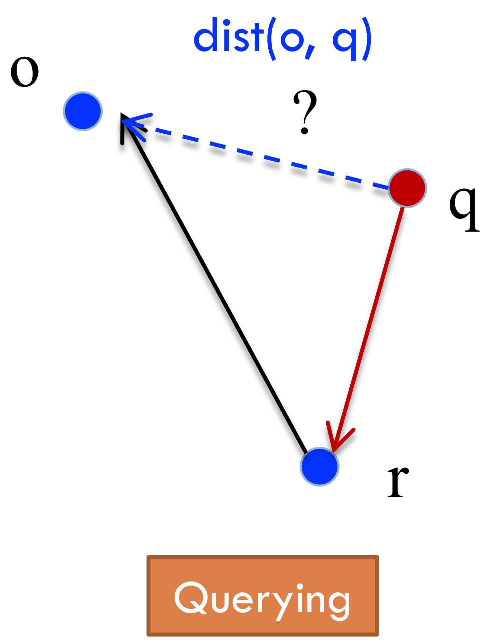
- Based on the metric property
  - (M)VP-tree, M-tree
- Based on intrinsic dimensionality
  - Cover Tree
- "Spill"
  - Spill for data: Spill Tree
  - Spill for query: Virtual Spill Tree

Able to index objects in a non-Euclidean space

Metric Property

- Inference on the lower & upper bound of dist(u, v)
  - Triangular inequality
  - Ptolemaic inequality

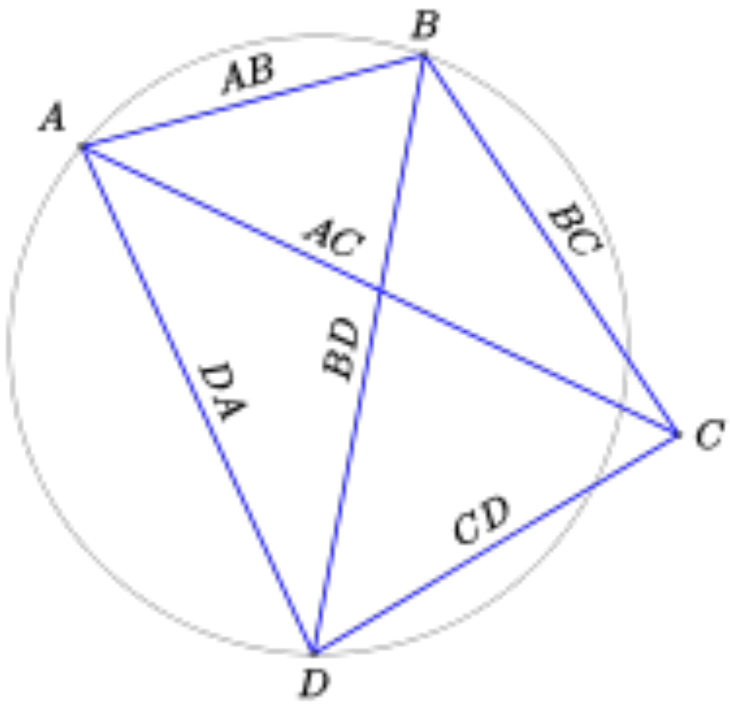




Triangular inequality:
- Lower and upper bounds of dist(o, q)

c.f., LSH (later)
- gives the full distribution of dist(o, q)

Ptolemaic inequality



$$\overline{AB} \cdot \overline{CD} + \overline{BC} \cdot \overline{DA} \geq \overline{AC} \cdot \overline{BD}$$

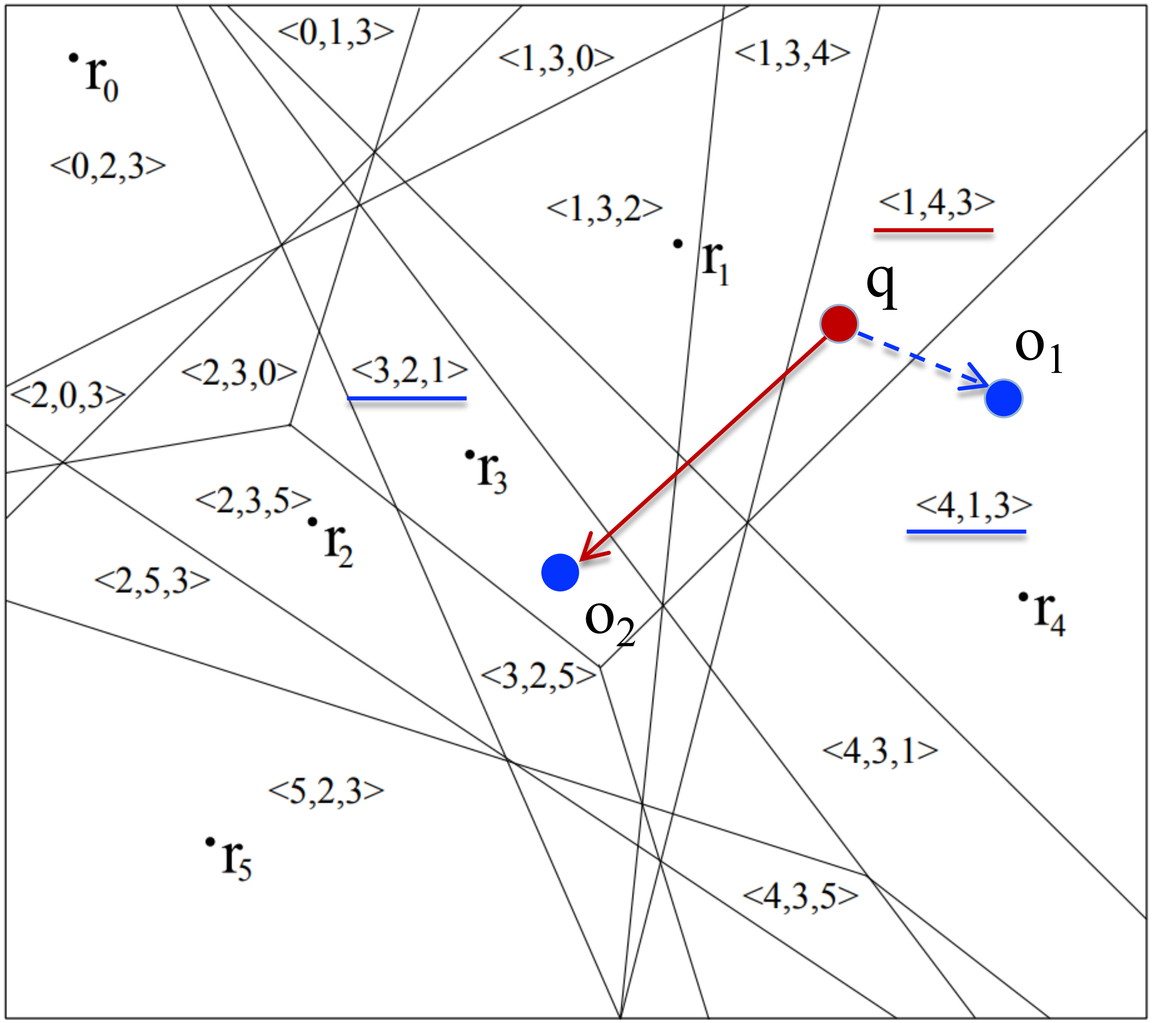
https://en.wikipedia.org/wiki/Ptolemy%27s_inequality

Variants

- Reference points
  - All DB objects: AESA
    - Organized into a hierarchical fashion → metric tree indexes
  - Many work/heuristics to select a good subset
- [Diversion] Use rank() instead of dist() of reference points
  - Permutation index [NBN16, …]
  - **dist**(u, v) is small → **d**(perm(u), perm(v)) is also small

PP-Index

Order-3 Voronoi Diagram

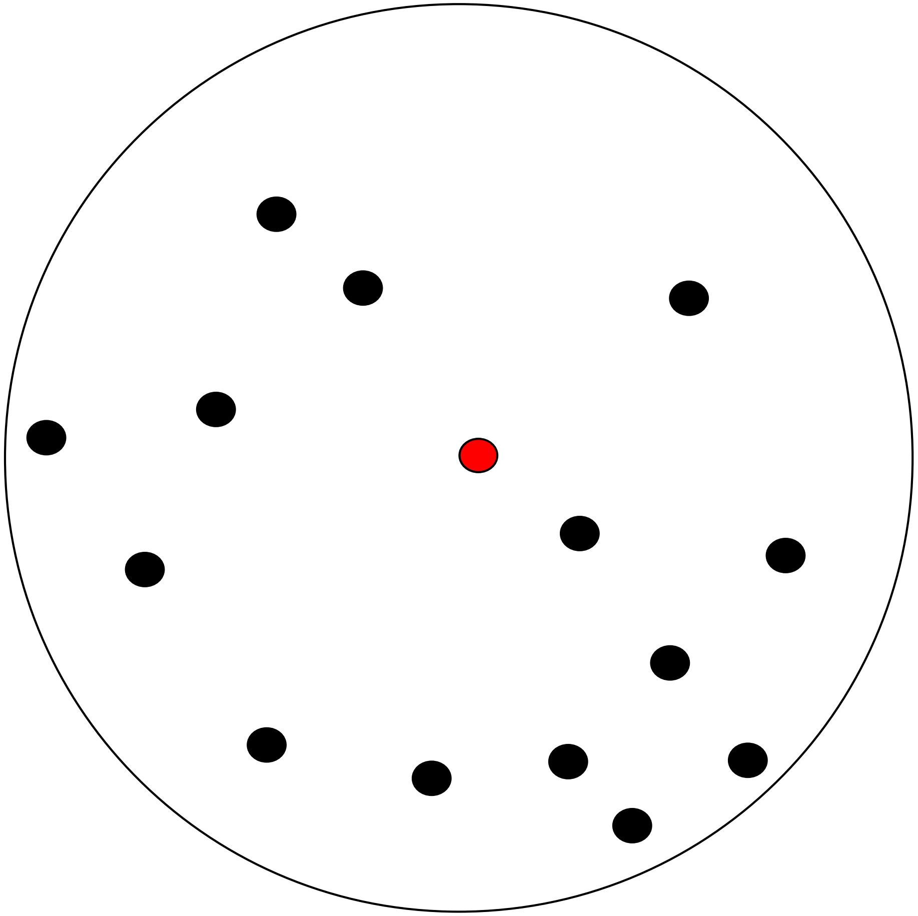


http://www.esuli.it/publications/PP-Index-slides.pdf

Intrinsic Dimensionality

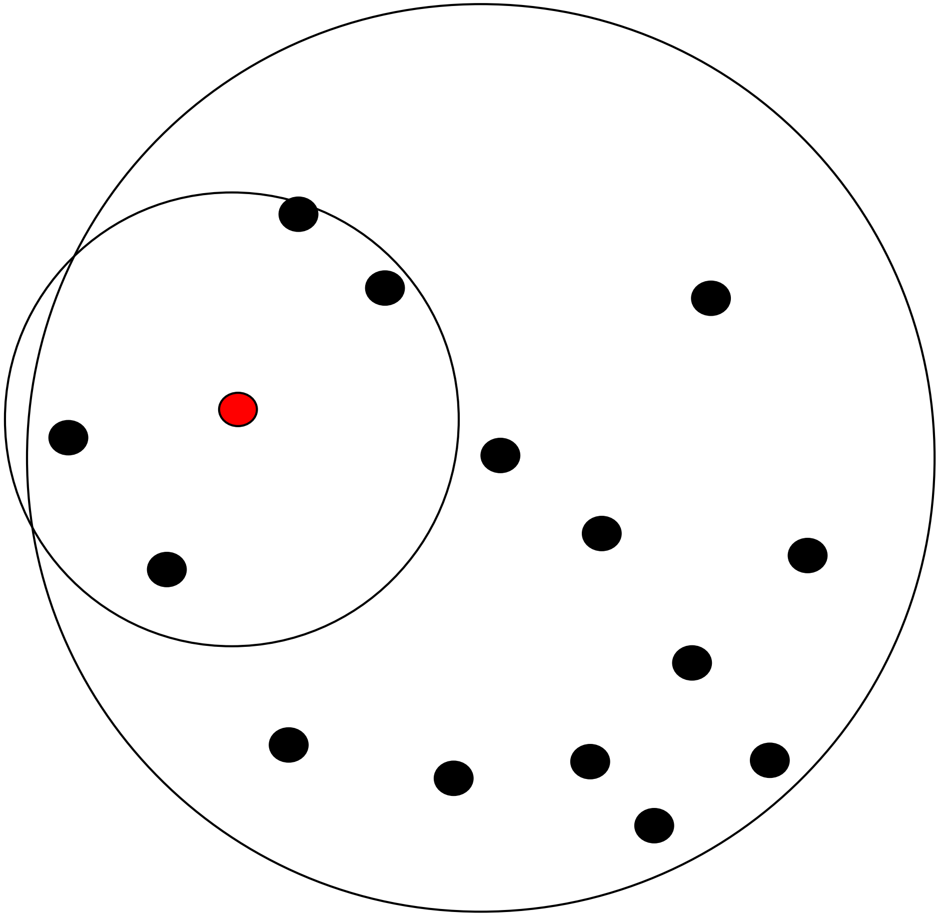
- One of the metrics is Expansion Constant
  - Smallest c such that |Ball(z, 2R)| ≤ c*|Ball(z, R)|, ∀z
- Cover Tree
  - O(n) space
  - O(c^6 * nlog(n)) construction and update time
  - O(c^{12} * ~~n~~log(n)) exact NN query time
  - $c^{O(1)} \log \Delta + (1/\epsilon)^{O(\log c)}$ ε-NN query
    - Δ (aspect ratio): ratio between largest and smallest interpoint distance

Cover Tree



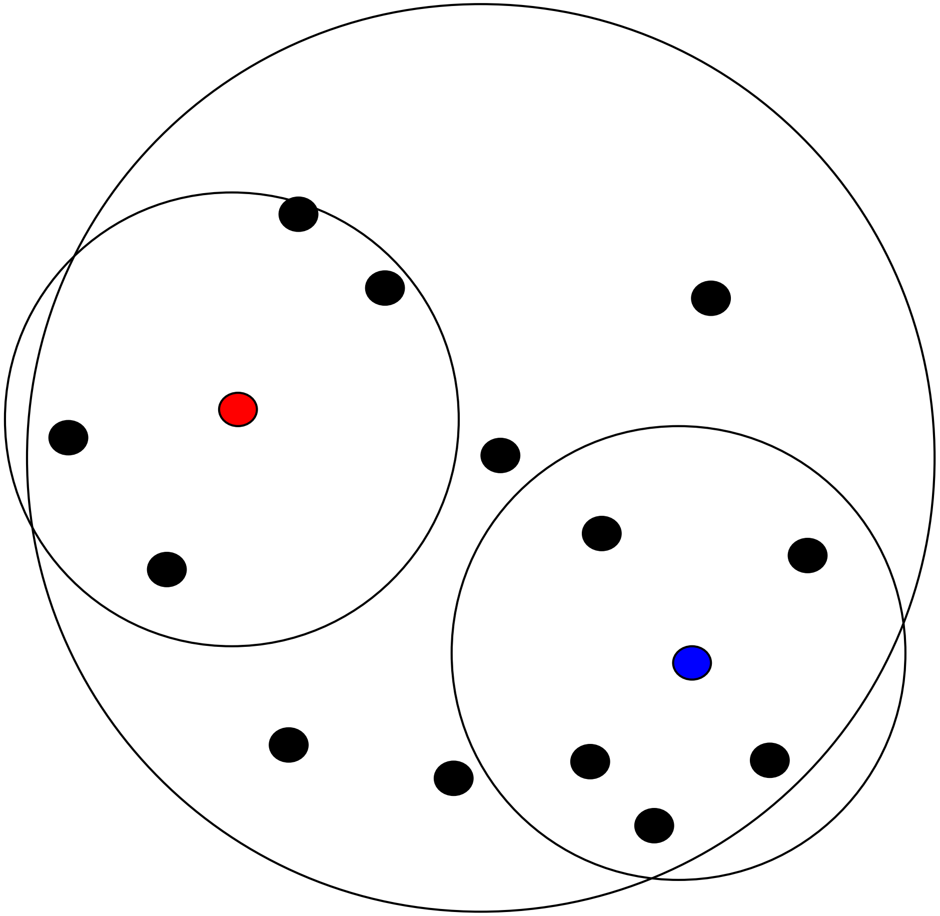
- A node covered by a pivot data point (red) with radius R

Cover Tree



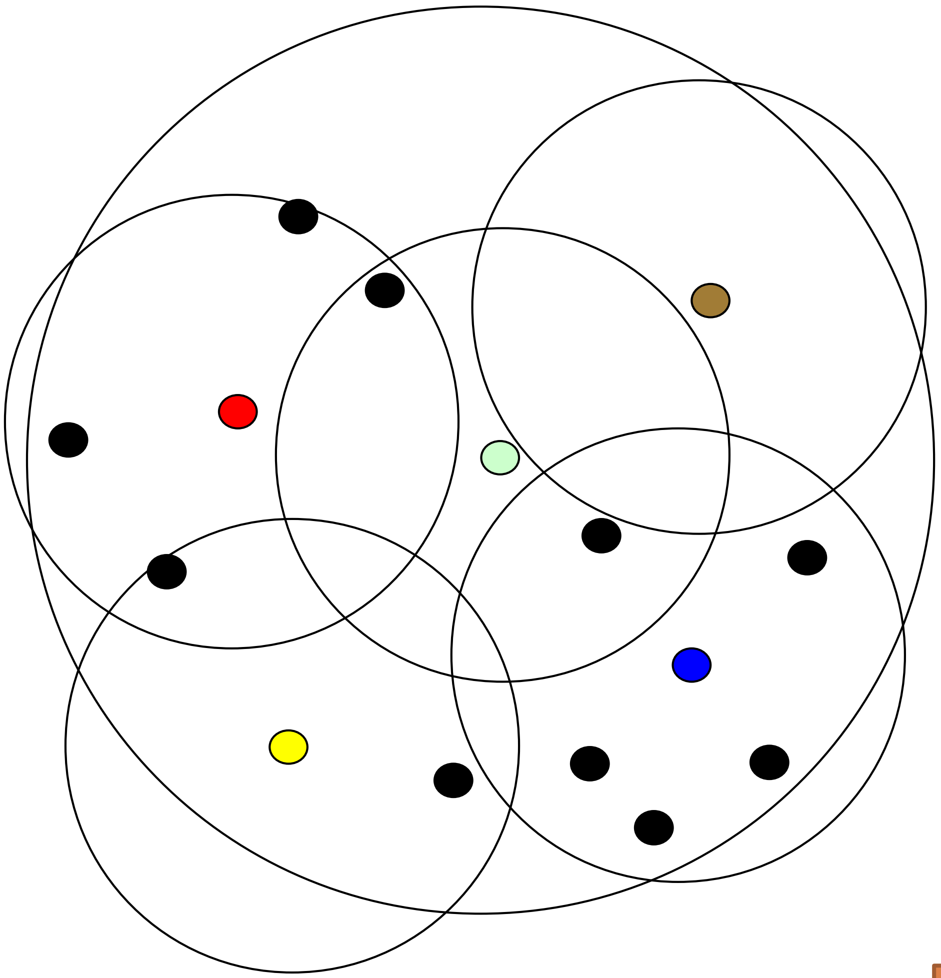
- Cover the points using a child pivot with radius R/2

Cover Tree



- Repeat by picking the child pivot outside the previous covers

Cover Tree



- Nesting
  - $C^{(i)}$: $C^{(i-1)}$ U black nodes
  - $C^{(i-1)}$: colored nodes
- Covering
  - $dist(u^{(i)}, v^{(i-1)}) \leq 2^i$
- Separation
  - $dist(u^{(i-1)}, v^{(i-1)}) \geq 2^i$

fan-out of any node $\leq c^4$

Encoding-based

- Learning to hash
- Product Quantization
- Hierarchical k-means

Learning to Hash

- Idea:
  - Embed R^d to a k-dimensional Hamming cube while minimizing some objective function (neighborhood preservation or distance distortion)
    - $x_i \in R^d \rightarrow z_i \in \{0, 1\}^k$

→ Partition the space into 2^k regions

- E.g., Spectral hashing:
  - Minimize $\sum_{ij} W_{ij} \|z_i - z_j\|$
    - and other conditions (max utilization of bits + uncorrelatedness)
  - Where $W_{ij} = \exp(-\|x_i - x_j\|^2 / \varepsilon^2)$

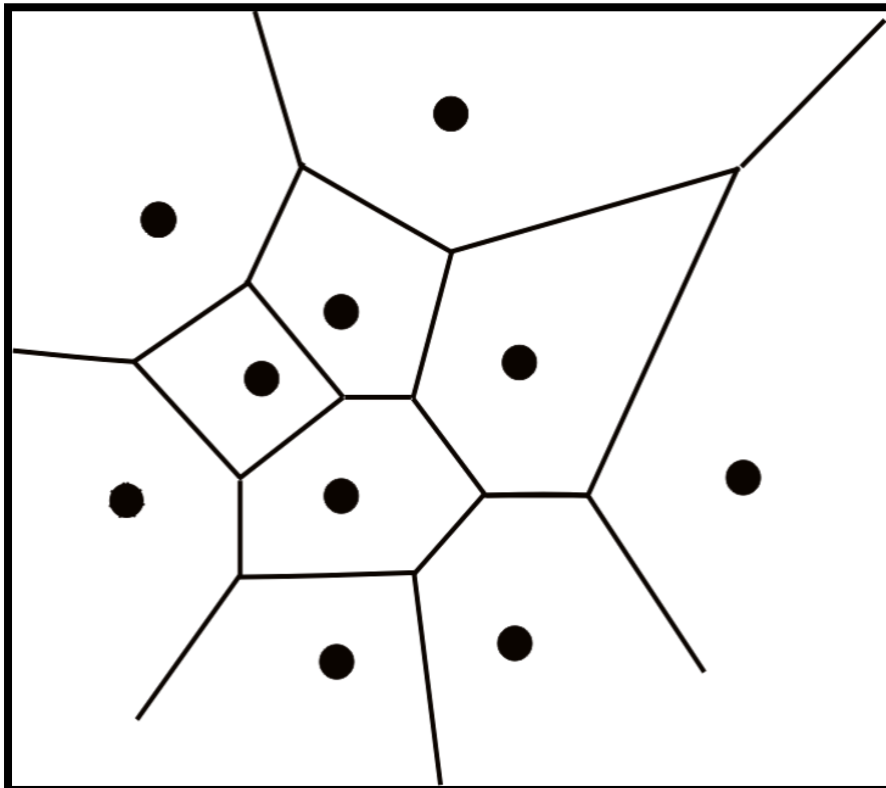
Minimize avg Hamming distance between neighboring points

- Many other variants

c.f., https://learning2hash.github.io and https://cs.nju.edu.cn/lwj/L2H.html

Coding based on k-means

- Partition the whole space into n regions by n-means → Voronoi
- Can be relaxed using k < n
  - However, still cannot afford a very large k (*why?*)



Solution 1: PQ (Product Quantization)

Tiny space consumption: ~1/32 size of the data

if $k = 2^8$

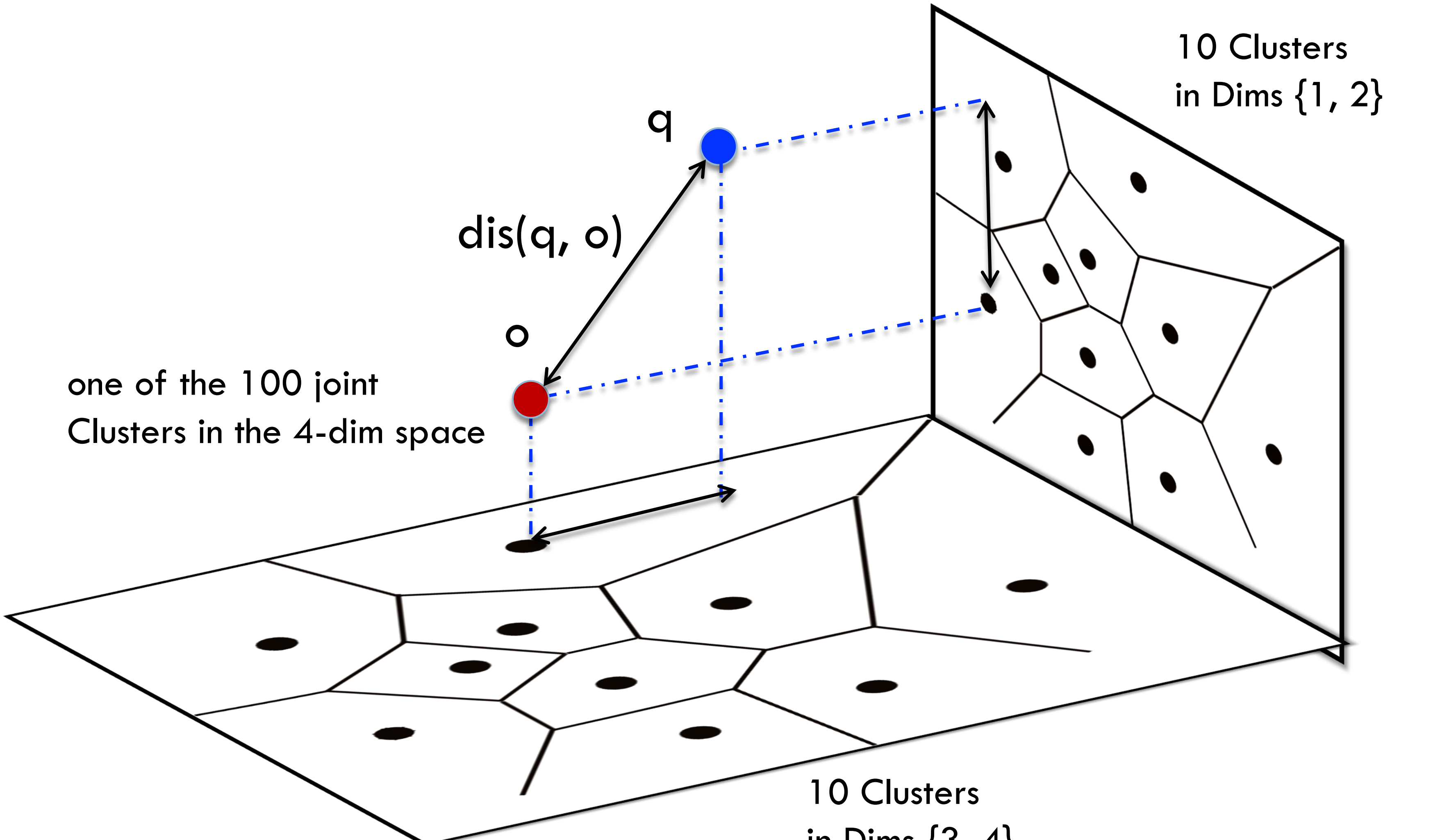
- Index:
  - Partition the d dims into L partitions
  - k-means clustering within each partition
  - $\{C_{1,j}\}$ X $\{C_{2,j}\}$ X … $\{C_{L,j}\}$ joint centers
  - Each point encoded as the closest joint center

Product

Quantization

- Query Processing:
  - Repeat
    - Find the closest joint center
    - Compute the asymmetric distance (via table lookup)
  - Optimizations:
    - Multi-index-based (best with only 2 partitions)
    - PQ Fast Scan [AKS15], PQBF [LCC17]

Illustration of PQ



Comparisons

	VA-File	PQ
#Partitions on dimensions	d	L = d/log(k)
Codebook	typically linear, equi-width partitioning of the domain	non-linear, "equi-width" partitioning of the domain
Query Processing	Brute-force on the encoded data	Best-first search on the encoded data

Solution 2: Hierarchical k-Means Tree

- PQ can be deemed as an approximate version of (L*k)-means quantization
- Hierarchical k-Means Tree (as in FLANN) recursively partition the data using k-means clustering using a small k
  - Special case: hierarchical 2-means trees

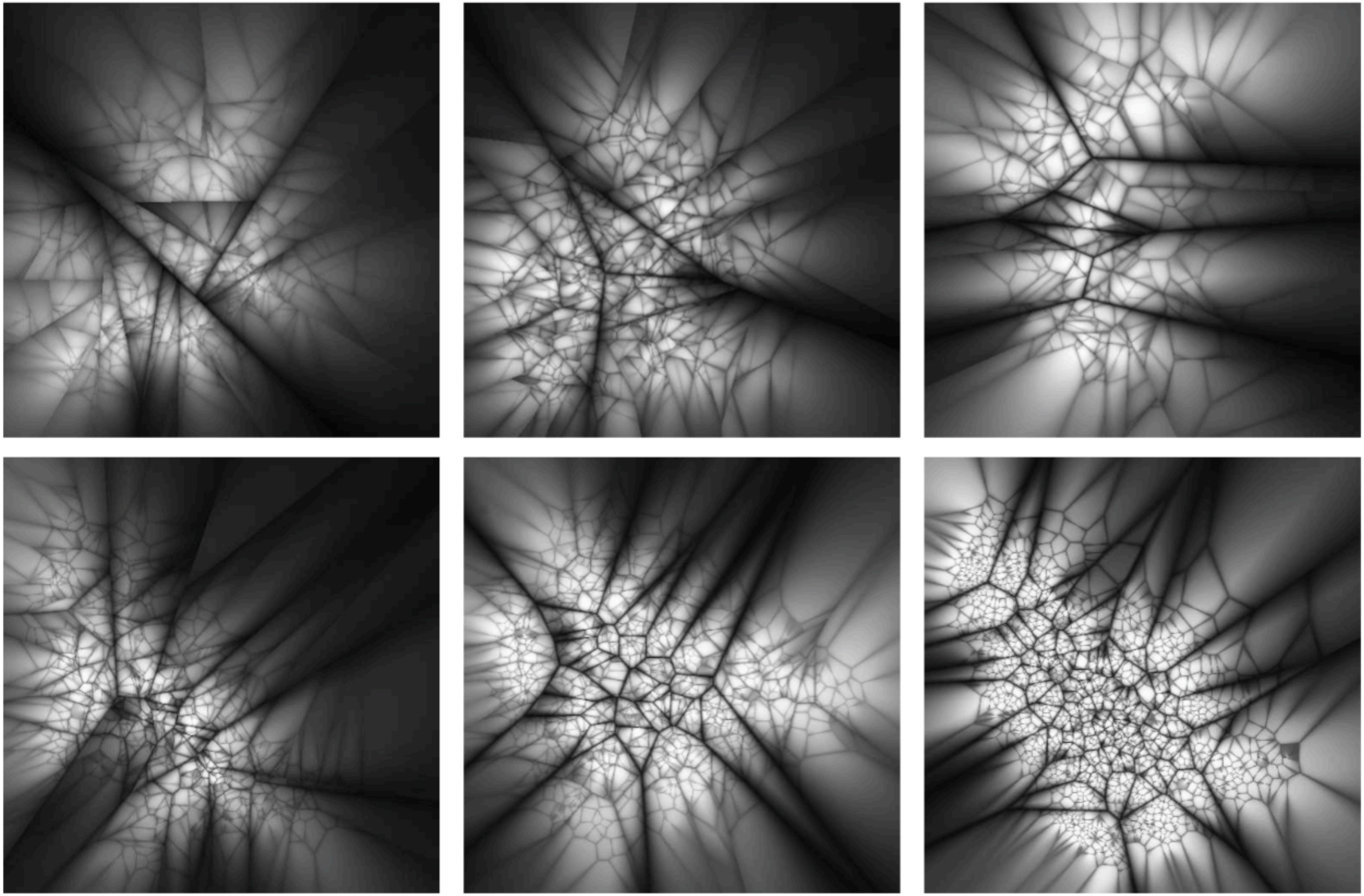


Figure 1: Projections of hierarchical k-means trees constructed using the same 100K SIFT features dataset with different branching factors: 2, 4, 8, 16, 32, 128. The projections are constructed using the same technique as in (Schindler et al., 2007). The gray values indicate the ratio between the distances to the nearest and the second-nearest cluster center at each tree level, so that the darkest values (ratio≈1) fall near the boundaries between k-means regions.