Indexing for Similarity Search

Qin Lv

COS598E Spring 2005

Outline

- Problem
- High-dimensional indexing: previous study
  – Quadtree, kd-tree, R-tree, …
- Locality sensitive hashing (LSH)
- Navigating nets
- Cover trees
- Conclusion

Problem

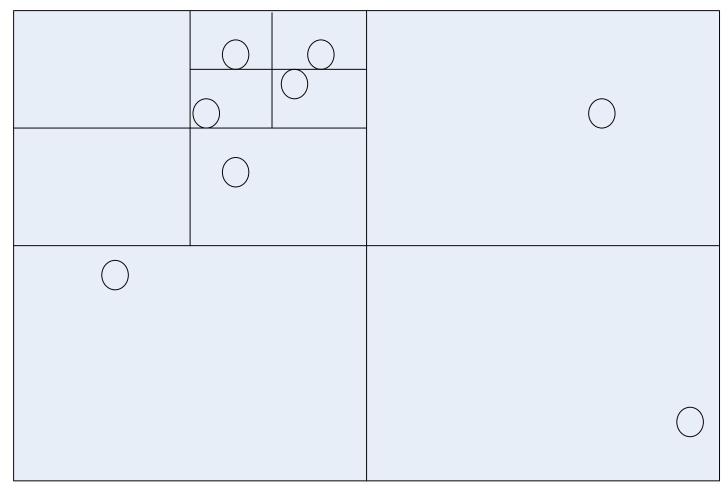
- Feature vectors: points in high-dimensional space
- Range query
- Nearest neighbor query
  - K-nearest neighbor query
  - Approximate nearest neighbor query
- Linear scan
- Indexing: preprocess/organize data points in order to answer queries efficiently

Reference: Indexing Survey

- Searching in high-dimensional spaces - Index structures for improving the performance of multimedia databases
  - Christian Böhm, Stefan Berchtold, Daniel A. Keim
  - ACM Computing Surveys (CSUR)
  - Volume 33 , Issue 3, Pages: 322 - 373
  - September 2001

Quadtree

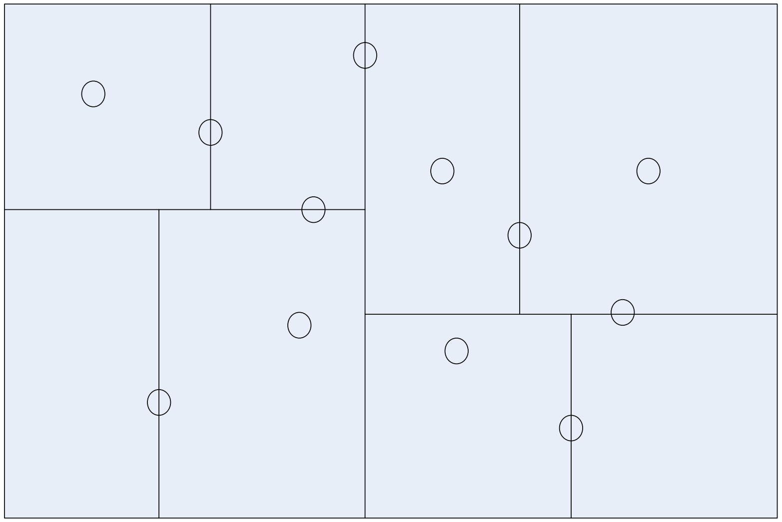
- At each level, splits a space into 2^d equal subspaces
  - 4 subspaces in 2-dimensional space, hence the name
- Very simple data structure
- But
  - Empty spaces
  - Space exponential in *d*
  - Time exponential in *d*



k-d Tree

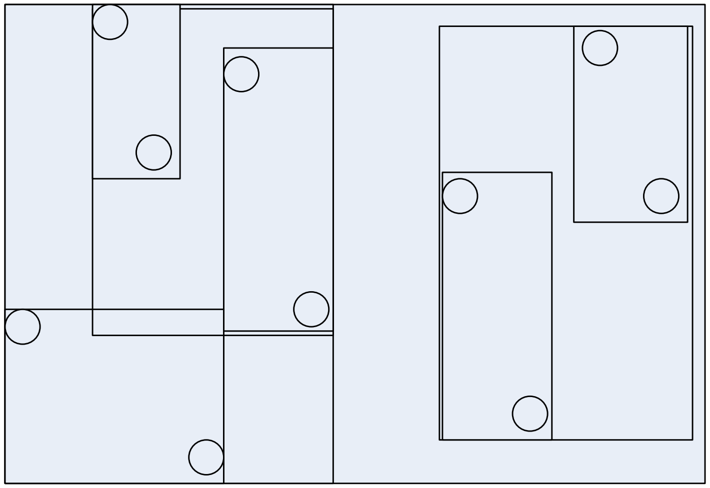
- Splits in one dimension each time
- Adaptive: instead of splitting in the middle, choose the split carefully
- No (or less) empty spaces
- Space linear to *d*
- Exponential query time

  still possible



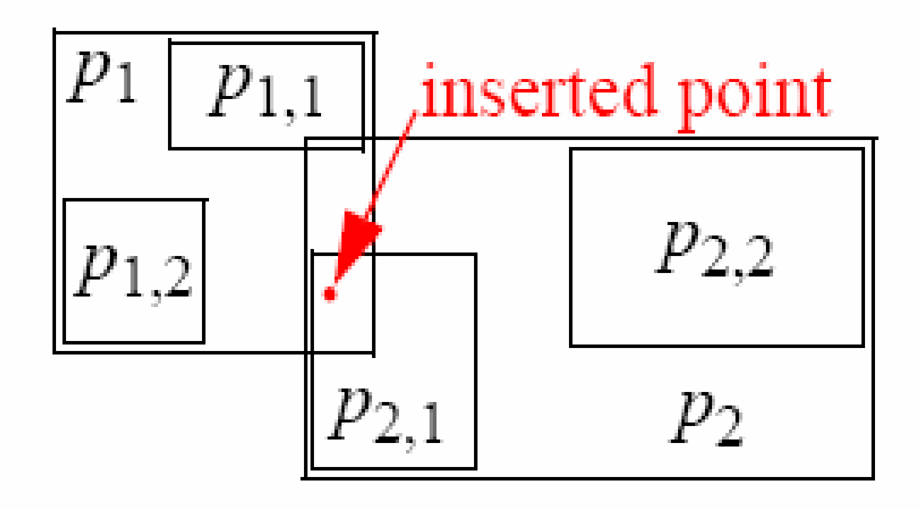
R-tree

- [Gut 84] Splits space using minimum bounding rectangles (MBRs)
- Insertion: starting from root, each time picks a child region, splits a region when needed
- Rules:
  - p is contained in exactly one region
  - p is contained in multiple regions
  - No region contains p



R-tree (cnt'd)

- Basic problem:
  - Overlap at high index levels and propagates down by misled insert operations

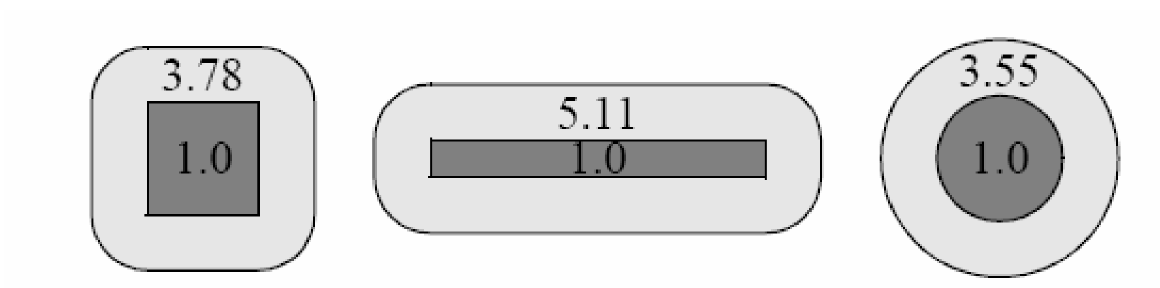


R*-tree

- [BKSS 90] an extension of R-tree
- *Minimize overlap between regions*
  - Picks the data region that yields the smallest enlargement of overlap
  - Picks the split plane that minimizes the overlap between regions
- *Minimize the surface of regions*
  - When splitting, picks the dimension that yields the smallest surface areas of all MBRs
- *Minimize the volume covered by internal nodes*
  - Picks the internal region that yields the smallest volume enlargement
- *Maximize the storage utilization*
  - *forced re-insert* : certain percentage of points with the largest distances from the region center are deleted and re-inserted

R*-tree (cnt'd)

- 10% - 75% improvements over R-tree
- In higher-dimensional spaces
  - Deteriorated directory (internal nodes)
  - Needs to load the entire index in order to process most queries
- Heuristics to optimize for regions with smaller surface is beneficial

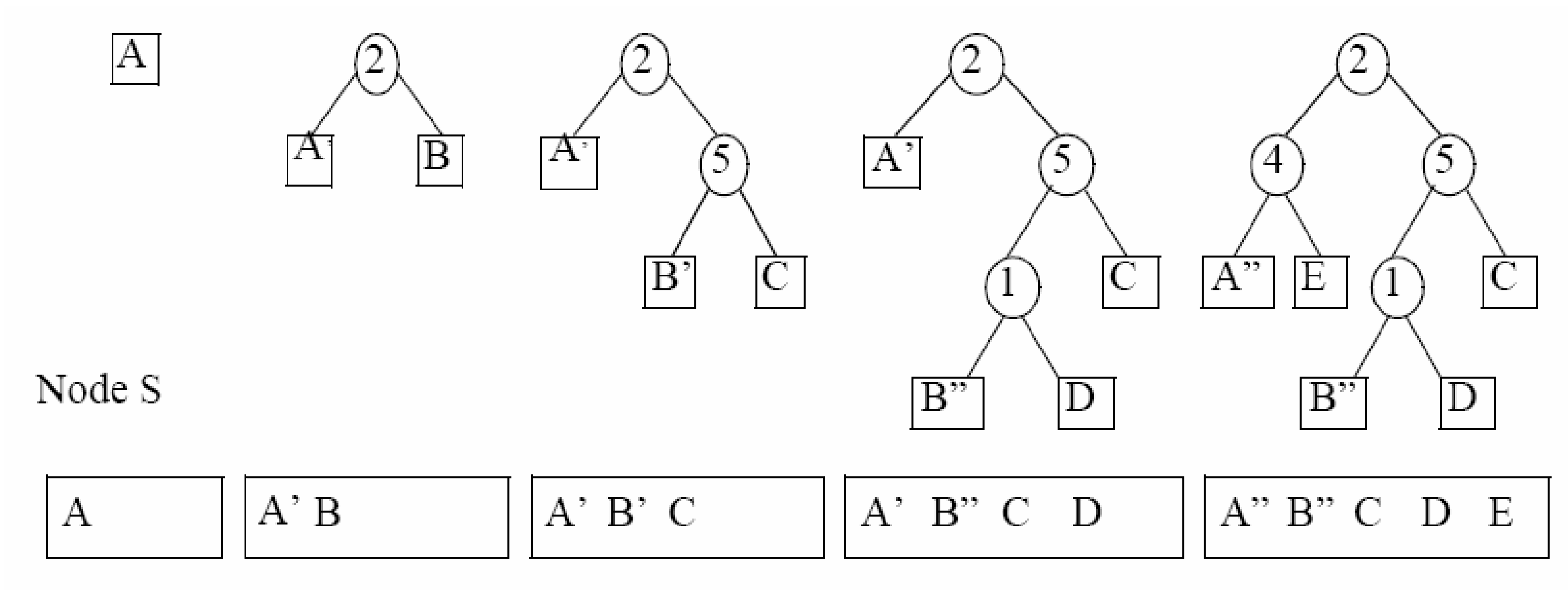


R^+-tree

- [SSH 86; SRF 87]
- An overlap-free variant of R-tree
- Uses forced-split to avoid overlap
- High dimensionality leads to many forced split operations
- Low storage utilization

X-tree

- [BKK 96] An extension of R*-tree
- Overlap-free split according to a split history
- Supernodes with enlarged page capacity



X-tree (cnt'd)

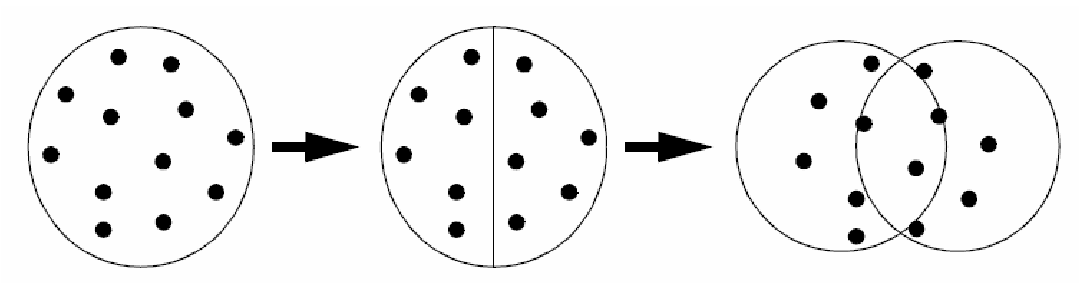
- Small dimensions
  - similar performance to R-tree
- Medium dimensions
  - high performance gain compared to R*-tree for all query types
- High dimensions
  - Also needs to visit large number of nodes
  - Linear scan is less expensive

SS-tree

- [WJ 96] uses spheres as regions
  – (centroid point, minimum radius)
- Insertion: at each level, chooses the child sphere whose centroid is closest to p
- *Forced re-insert*: 30% points with largest distances to centroid are deleted and re-inserted

SS-tree (cnt'd)

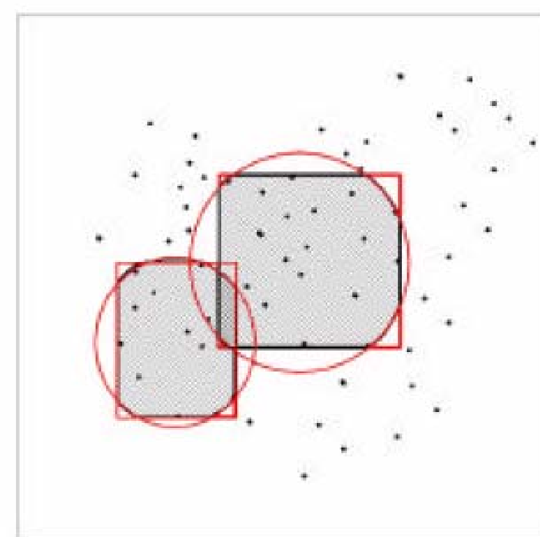
- Although spheres are theoretically superior to volume-equivalent MRBs
  - Overlap-free split is difficult for spheres



- Performance
  - Outperforms R*-tree by a factor of 2
  - Not as good as LSD^h-tree and X-tree

SR-tree

- [KS 97] A combination of R*-tree and SS-tree
  - Region: intersection between a rectangle and a sphere
  - 2d values for MBRs
  - d+1 values for spheres
- Insert and split operations similar to SS-tree

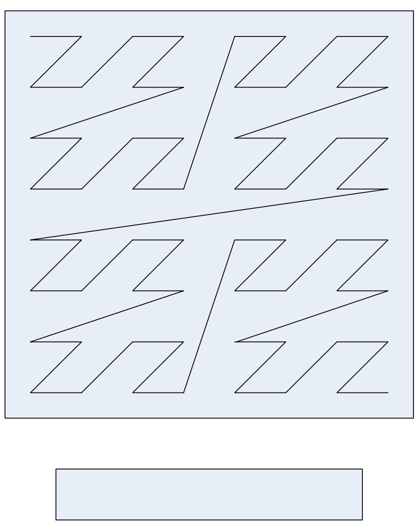
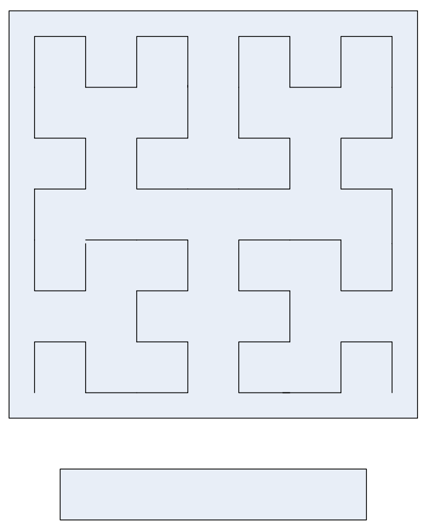
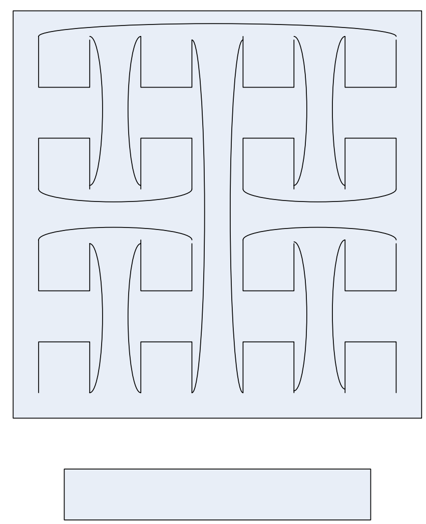


SR-tree (cnt'd)

- Reports better performance than SS-tree and R*-tree
- Probably not as good as LSD^h-tree and X-tree

Space Filling Curves

- Mappings from *d*-dimensional space to one-dimensional space
- Points that are close in original space are likely to be close in the embedded space
- Embedded space can be indexed by B-tree

Name	Region	Disjoint	Complete	Criteria for Insert	Criteria for Split	Re-insert
R-tree	MBR	No	No	Volume enlargement Volume	Various algorithms	no
R*-tree	MBR	No	No	Overlap enlargement Volume enlargement Volume	Surface area Overlap Dead space coverage	Yes
X-tree	MBR	No	No	Overlap enlargement Volume enlargement Volume	Split history Surface/overlap Dead space coverage	no
LSD^h-tree	K-d-tree region	Yes	No/yes	Unique due to complete, disjoint partition	Cyclic change of dim. # of distinct values	no
SS-tree	Sphere	No	No	Proximity to centroid	Variance	yes
SR-tree	Intersect sphere/ MBR	No	No	Proximity to centroid	Variance	yes
Space filling curves	Union of rectangle s	Yes	Yes	Unique due to complete, disjoint partition	According to space filling curve	no

Locality Sensitive Hashing (LSH)

- (*r1,r2, p1, p2*)-sensitive hashing
- *L* hash tables
- Each hash table examines *k* random bits

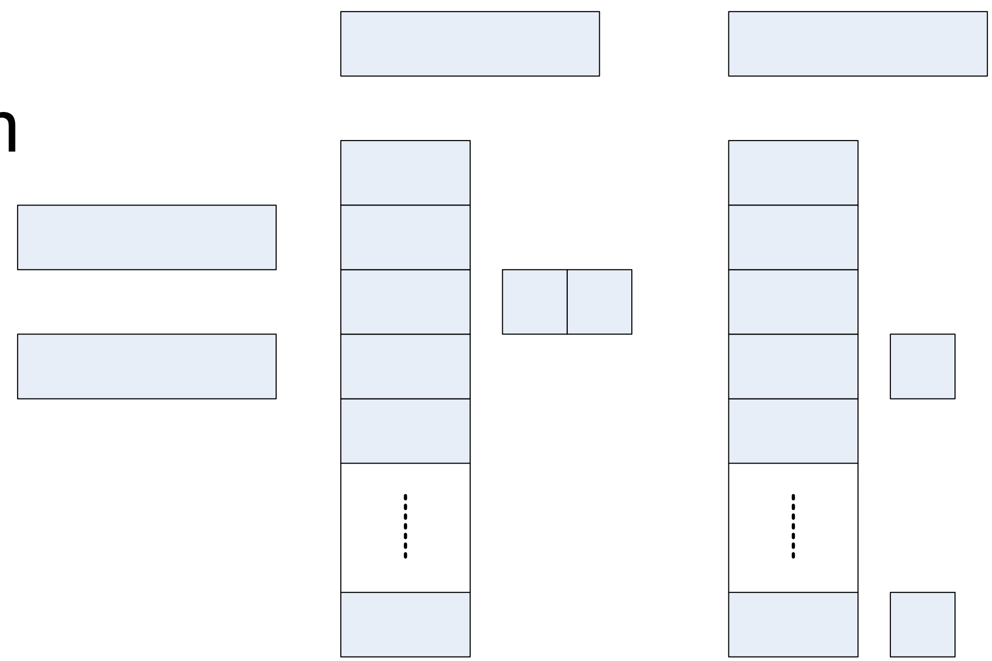
$$if\ \ p \in B(q, r_1)\ then\ \Pr_H[h(q) = h(p)] \geq p_1$$

$$if\ \ p \notin B(q, r_2)\ then\ \Pr_H[h(q) = h(p)] \leq p_2$$

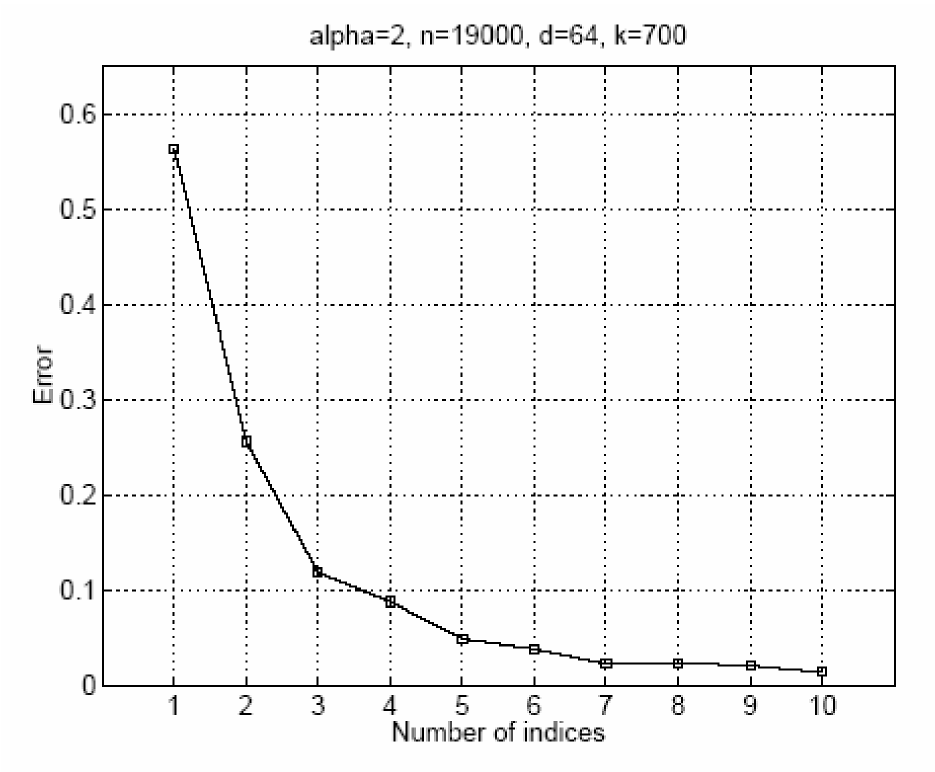
$$\rho = \frac{\ln 1/p_1}{\ln 1/p_2}$$

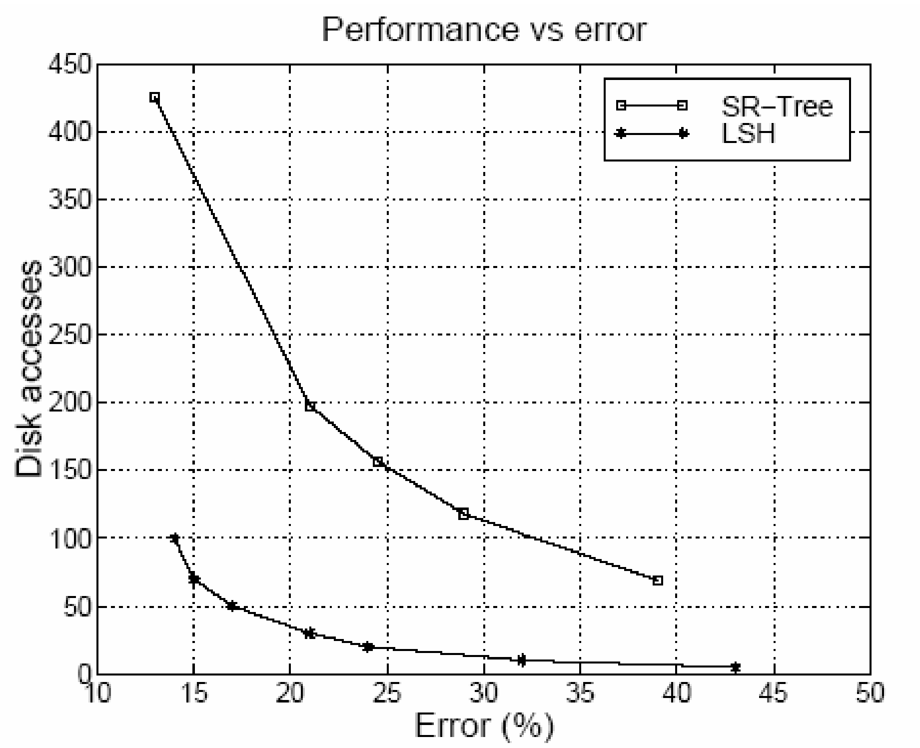
$$k = \log_{1/p_2}(n/B)$$

$$l = (n/B)^{\rho}$$

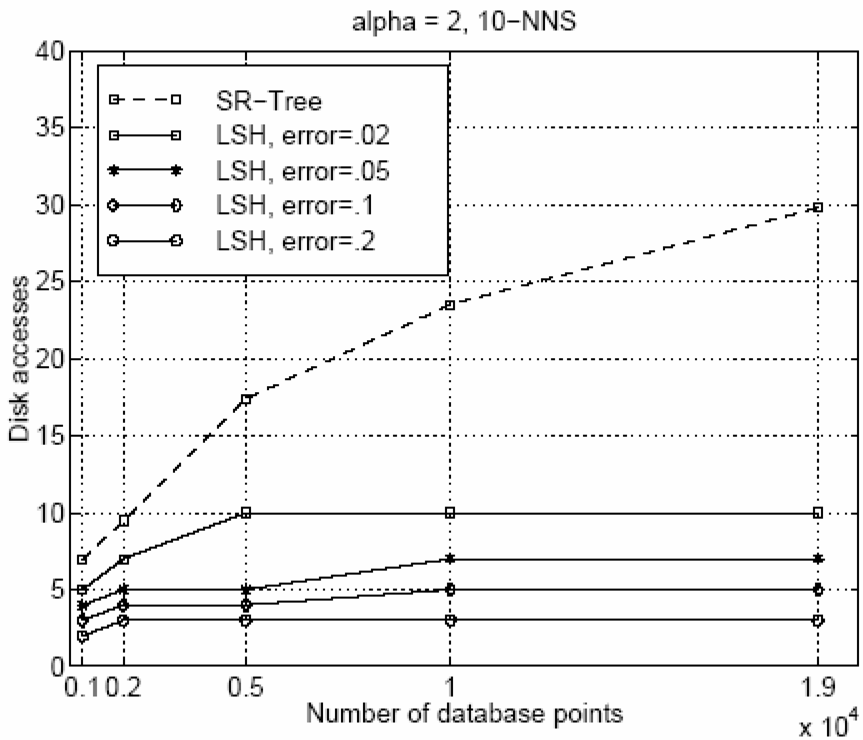


LSH Performance

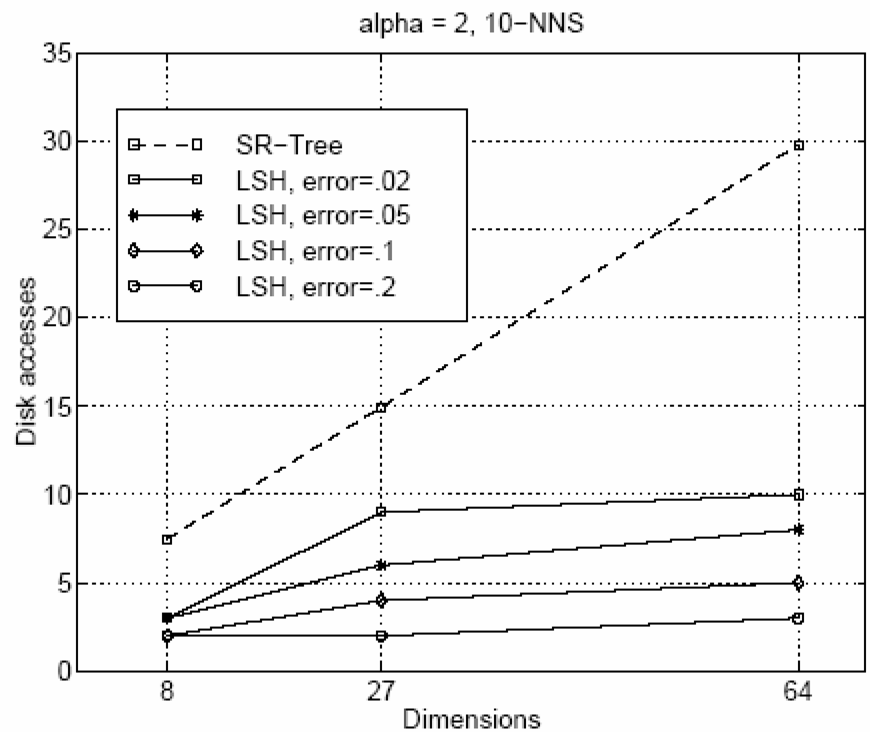




LSH Performance (cnt'd)



(b) Approximate 10-NNS



Intrinsic Dimensionality

- Metric space (X, d)
- Closed ball $B_S(x,r) = \{y \in S : d(x,y) \le r\}$
- *Doubling dimension*
  - Minimum value *ρ* such that every set in X can be covered by *2^ρ* sets of half the diameter
- *Expansion constant*
  - Smallest value c ≥ 2 such that

    $$|B_S(p,2r)| \le c\,|B_S(p,r)|$$

Navigating Nets

- Leveled directed acyclic graph
  - Multiple paths may exist from top to a lower-level point
- Each consequent level “covers” the dataset on a finer scale
- Adjacent levels are connected by pointers allowing for navigation between scales

Navigating Nets

A subset $Y \subseteq X$ is an ε - *net* of X if it satisfies

(1) $\forall x, y \in Y, d(x, y) \geq \varepsilon$

(2) $X \subseteq \bigcup_{y \in Y} B(y, \varepsilon)$

Let $\Gamma = \{2^i : i \in Z\}, Y_r$ be a r - *net* of $Y_{r/2}$.

For every scale $r \in \Gamma$ and every $y \in Y$, the data structure stores a list of nearby points to y among the $r/2$ - *net* $Y_{r/2}$.

- *scale r navigation list of y* is defined by

$$L_{y,r} = \{z \in Y_{r/2} : d(z, y) \leq \gamma \cdot r\}$$

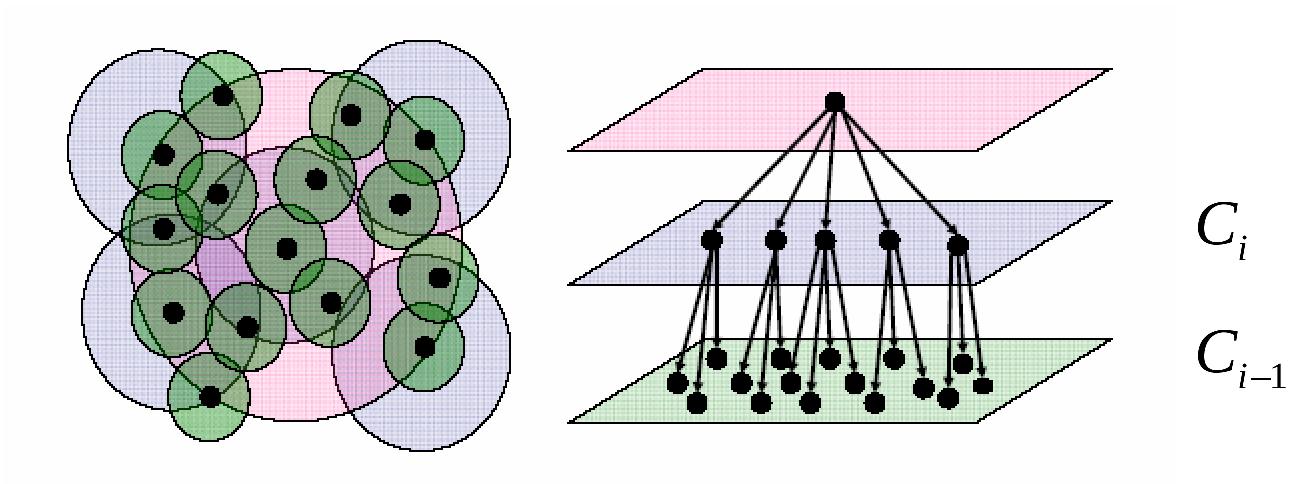
Navigating Nets: Query

$\text{APPROX-NNS}(\text{Input } q \in X \text{ and } \varepsilon > 0)$

1. set $r = r_{\max}$ and $Z_r = \{y_{top}\}$
2. while $2r(1+1/\varepsilon) > d(q, Z_r)$ and Z_r is proper
3. set $Z_{r/2} = \{y \in \bigcup_{z \in Z_r} L_{z,r} : d(q,y) \le d(q, Z_r) + r\}$
4. set $r = r/2$
5. return $z \in Z_r$ for which $d(q,z)$ is minimal

Cover Trees

- a leveled tree where each level is a "cover" for the level beneath it
  - *Nesting*: $C_i \subset C_{i-1}$
  - *Covering tree*: For every node $p \in C_{i-1}$, there exists a node $q \in C_i$ satisfying $d(p,q) \le 2^i$ and exactly one such *q* is a parent of *p*
  - *Separation*: For all nodes $p,q \in C_i$, $d(p,q) > 2^i$



Cover Trees: Incremental Construct

Insert (point p, covert set Q_i, level i)

(1) set $Q = \{\text{Children}(q) : q \in Q_i\}$

(2) if $d(p, Q) > 2^i$ then return "no parent found"

(3) else

(a) set $Q_{i-1} = \{q \in Q : d(p, q) \leq 2^i\}$

(b) if Insert $(p, Q_{i-1}, i-1) =$ "no parent found" and $d(p, Q_i) \leq 2^i$

(i) pick $q \in Q_i$ satisfying $d(p, q) \leq 2^i$

(ii) insert p into Children(q)

(iii) return "parent found"

(c) else return "no parent found"

Cover Trees: Query

Find - Nearest(cover tree T, query point p)

(1) set $Q_\infty = C_\infty$

(2) for i from ∞ down to $-\infty$

(a) consider the set of children of Q_i

$$Q = \{\text{Children}(q) : q \in Q_i\}$$

(b) form next cover set :

$$Q_{i-1} = \{q \in Q : d(p,q) \le d(p,Q) + 2^i\}$$

(3) return $\min_{q \in Q_{-\infty}} d(p,q)$

Cover Trees: Batch Construct

Construct(point p, point sets $<$ NEAR, FAR $>$, level i)

(1) if NEAR $= \Phi$

(2) then return $< p, \Phi >$

(3) else

(a) $<$ SELF, NEAR $> =$ Construct(p, SPLIT($d(p,\cdot), 2^{i-1}$, NEAR), $i-1$)

(b) add SELF to Children(p_i)

(c) while NEAR $\neq \Phi$

(i) pick q in NEAR

(ii) $<$ CHILD, UNUSED $>=$ Construct(q, SPLIT($d(q,\cdot), 2^{i-1}$, NEAR, FAR), $i-1$)

(iii) add CHILD to Children(p_i)

(iv) let $<$ NEW - NEAR, NEW - FAR $> =$ SPLIT($d(q,\cdot), 2^{i}$, UNUSED)

(v) add NEW - FAR to FAR and NEW - NEAR to NEAR

(d) return $< p_i$, FAR $>$

Cover Trees: Batch Query

$\text{Find-All-Nearest (query cover tree } p_j \text{, cover set } Q_i)$

(1) if $i = -\infty$ then for each $a \in L(p_j)$

return $\arg\min_{b \in Q_{-\infty}} d(a,b)$ as the nearest neighbor of a

(2) else

(a) if $j < i$ then

(i) Set $Q = \{\text{Children}(q) : q \in Q_i\}$

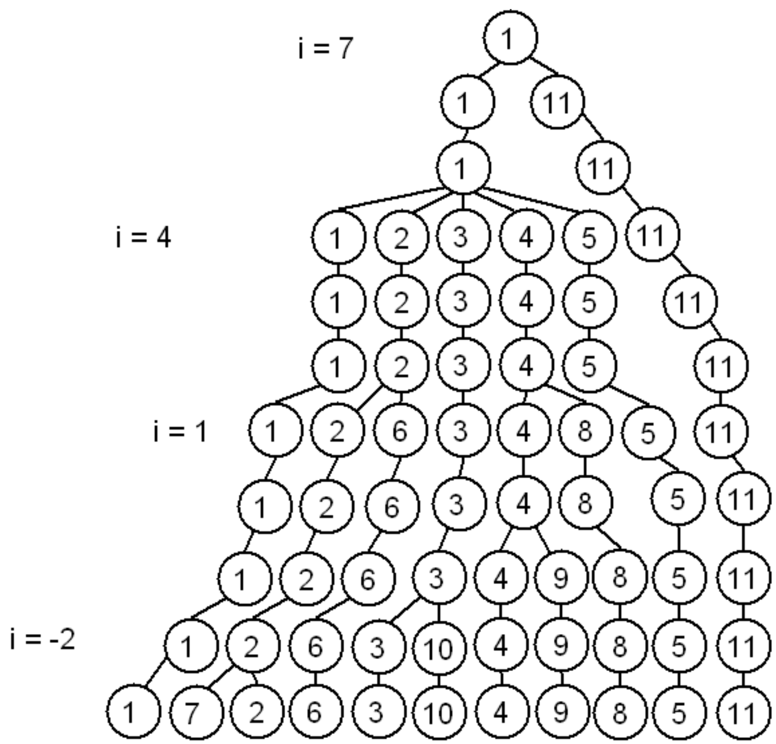
(ii) Set $Q_{i-1} = \{q \in Q : d(p_j, q) \le \min_{q \in Q} d(p_j, q) + 2^i + 2^{j+2}\}$

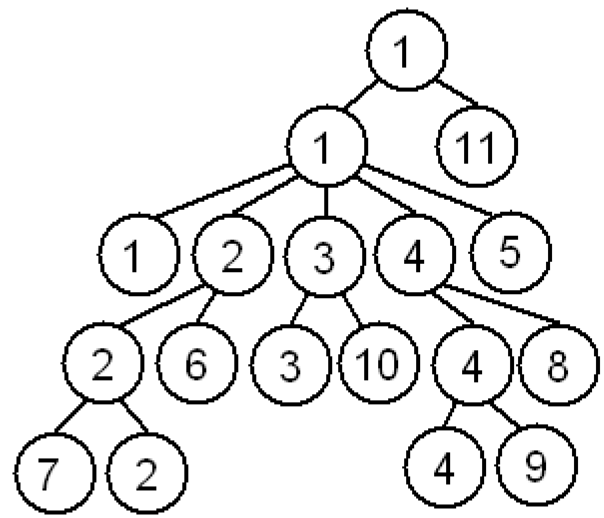
(iii) $\text{Find-All-Nearest}(p_j, Q_{i-1})$

(b) else for each $q_{j-1} \in \text{Children}(p_j)$

$\text{Find-All-Nearest}(q_{j-1}, Q_i)$

Cover Trees: Space Complexity





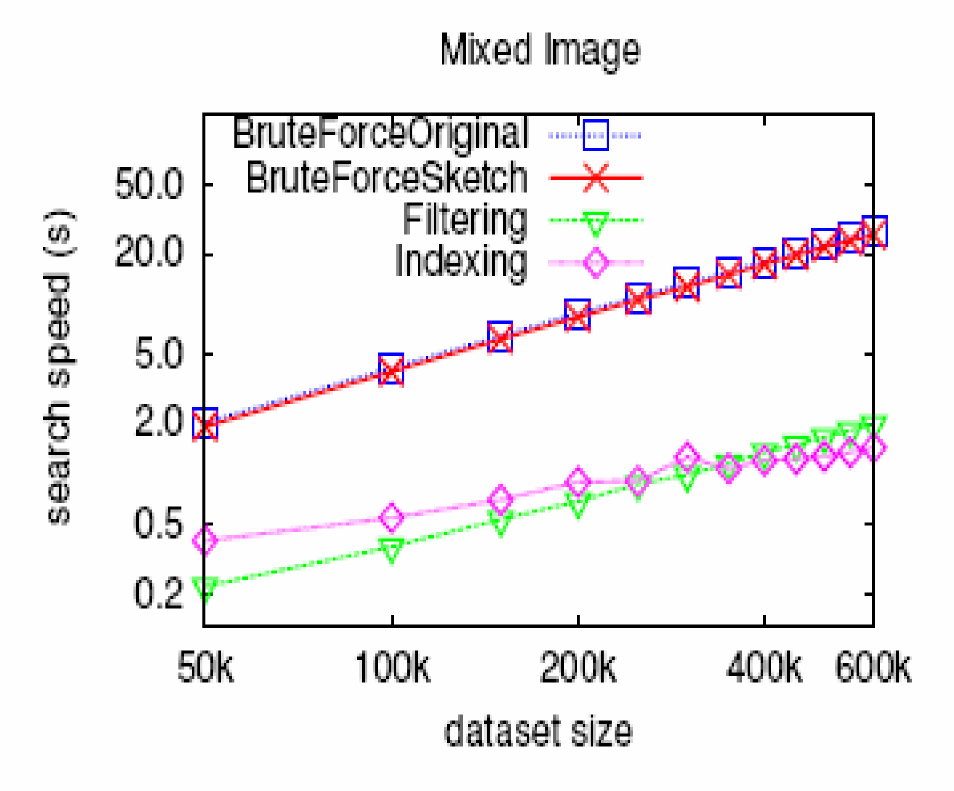
www.lems.brown.edu/vision/ independentStudy/Voctoria_covertree.ppt

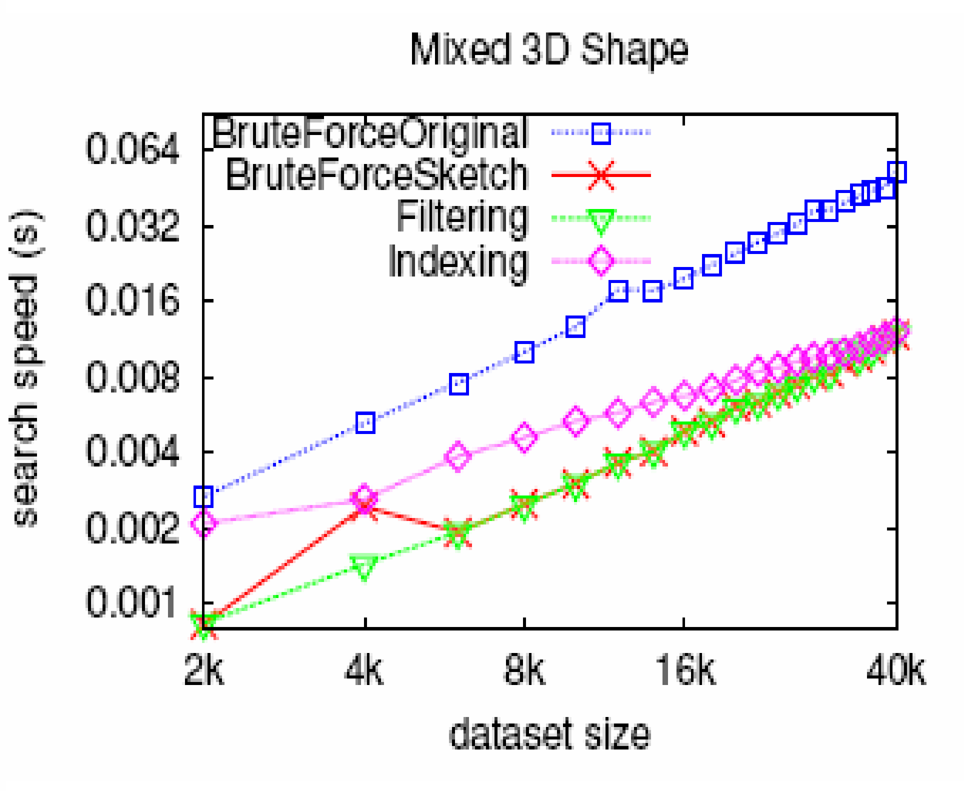
Cover Trees vs. Navigating Nets

	Cover Trees	Navigating Nets
Construction Space	$O(n)$	$c^{O(1)}n$
Construction Time	$O(c^6 n \ln n)$	$c^{O(1)}n \ln n$
Insertion/Removal	$O(c^6 \ln n)$	$c^{O(1)} \ln n$
Query	$O(c^{12} \ln n)$	$c^{O(1)} \ln n$
Batch Query	$O(c^{16} n)$	$c^{O(1)}n \ln n$

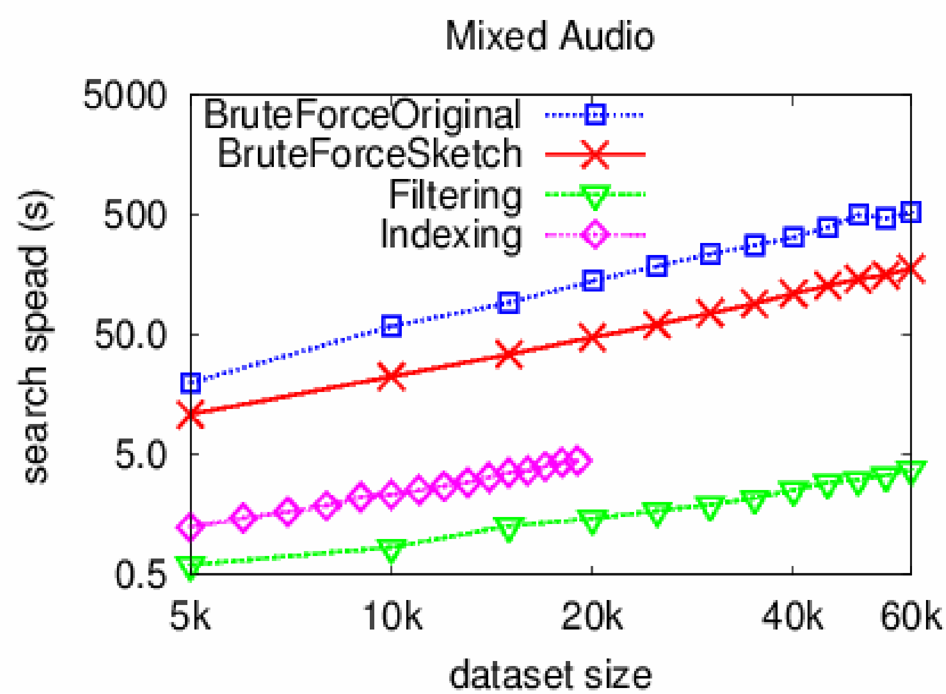
Dataset	d	N	Cover tree (s)	Brute force (s)	Speedup factor
Iris	4	150	0.0012	0.0014	1.2
Bupa	7	345	0.005	0.007	1.5
Wine	14	178	0.001	0.003	1.8
Glass	10	214	0.002	0.004	2.0
Pima	9	768	0.010	0.047	4.9
Ionosphere	35	351	0.006	0.017	3.0
Pendigits_A	15	3498	0.091	1.367	15.1
Optdigits_A	65	1797	0.277	0.811	3.0
Pendigits_B	15	7494	0.340	6.606	19.4
Optdigits_B	65	3823	1.493	3.872	2.6
Letter	17	20000	6.057	37.633	6.2
Corel	32	37749	38.569	203.2	5.3
Image	4096	698	0.871	1.208	1.4
Phy_train	78	50000	13.867	724.5	52.2
Phy_test	78	100000	38.924	2885.5	74.1
Bio_train	74	145751	814.4	6134.9	7.5
Bio_test	74	139658	741.6	5672.9	7.7
covtype	55	581012	77.9	72301.3	928.3
Mnist	784	60000	1944.0	4581.6	2.4

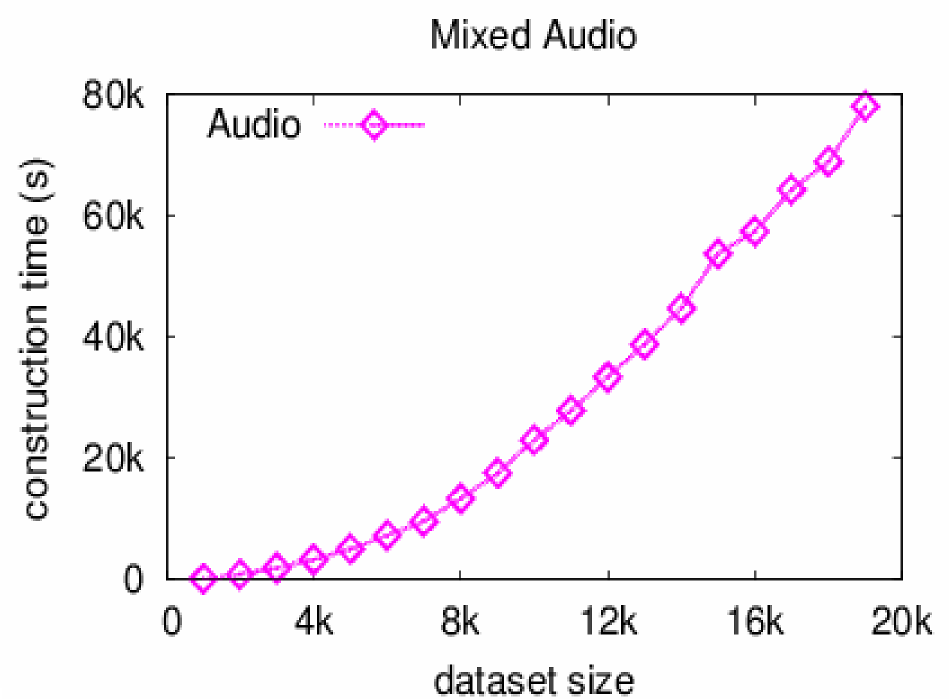
Cover Trees: Performance





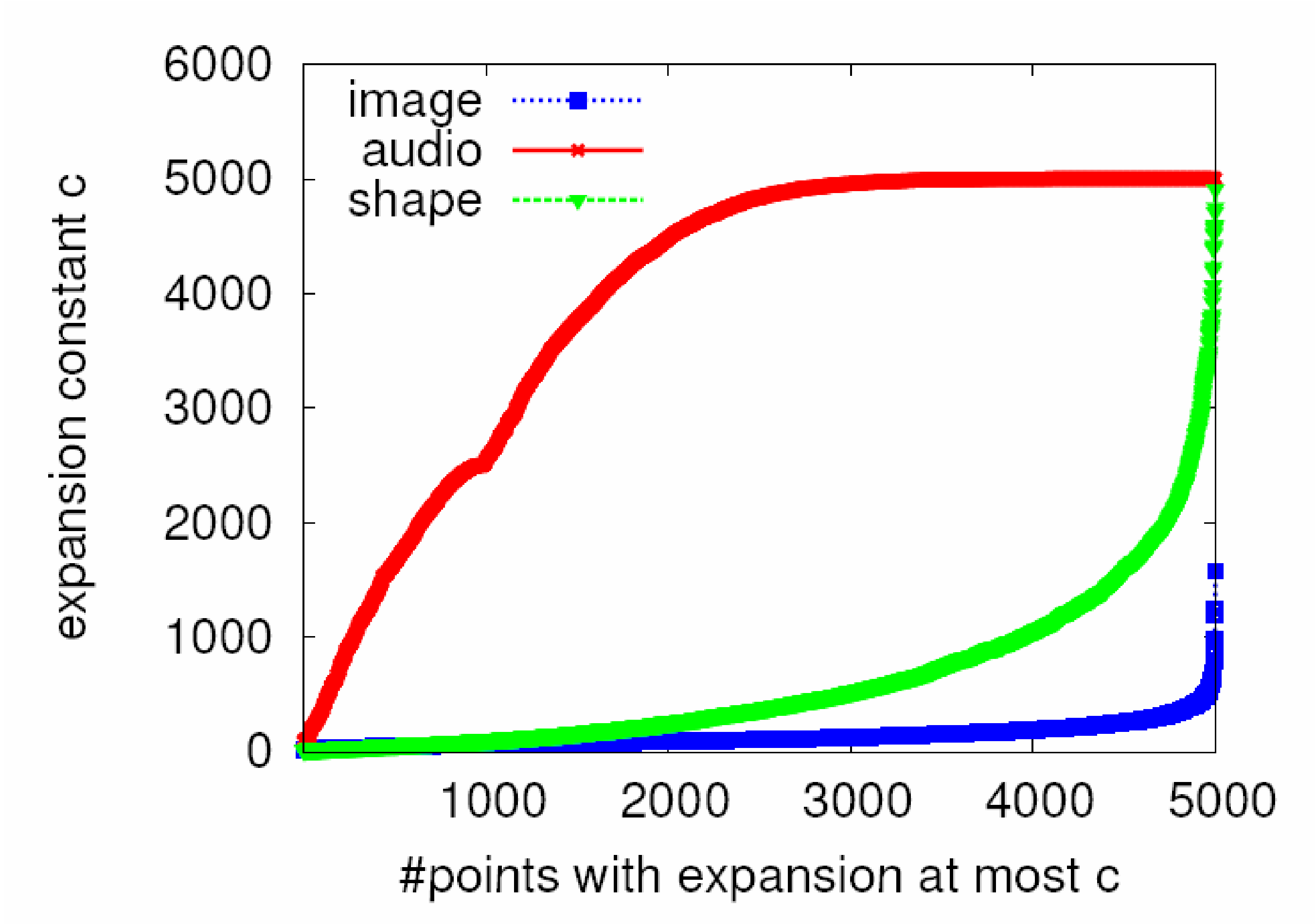
Cover Trees: Performance (cnt'd)





* Results from Zhe Wang

Expansion Constant



Doubling Dimension*

Audio

$Log_2(r)$	2^ρ	ρ
1	86	6.43
2	1992	10.96
3	1.38	0.47

Image

$Log_2(r)$	2^ρ	ρ
1	4	2
2	50.41	5.66
3	139.3	7.12
4	25.10	4.65
5	1.17	0.23
6	1.0004	0.0006

Shape

$Log_2(r)$	2^ρ	ρ
1	9	3.17
2	224.08	7.81
3	187.90	7.55
4	5.40	2.43
5	1.72	0.78
6	1.32	0.40
7	1.11	0.15
8	1.09	0.13
9	1.04	0.05
10	1.02	0.03
11	1.01	0.01
12	1.003	0.004
13	1.0004	0.0006
14	1.002	0.003
15	1.0009	0.0013

* Results from Emily Huang

Conclusion

- Indexing high-dimensional data for similarity search is hard
- Better feature vectors and distance functions
- A hybrid approach?
  – Cover trees
  – Locality sensitive hashing
  – Linear scan