Visual Object Recognition

Lecture 6: Tree Search and Hashing

Per-Erik Forssén, docent
Computer Vision Laboratory
Department of Electrical Engineering
Linköping University

Seminar 8 date

- All seminars have been shifted by one week (again).
- Time for LE8 is now April 7 12.30-15.

Lecture 6: Tree Search and Hashing

- High dimensional spaces
  Distances, Region edges

- Search Trees
  kD-Trees, Best Bin First (BBF), Ball Trees,K-means tree

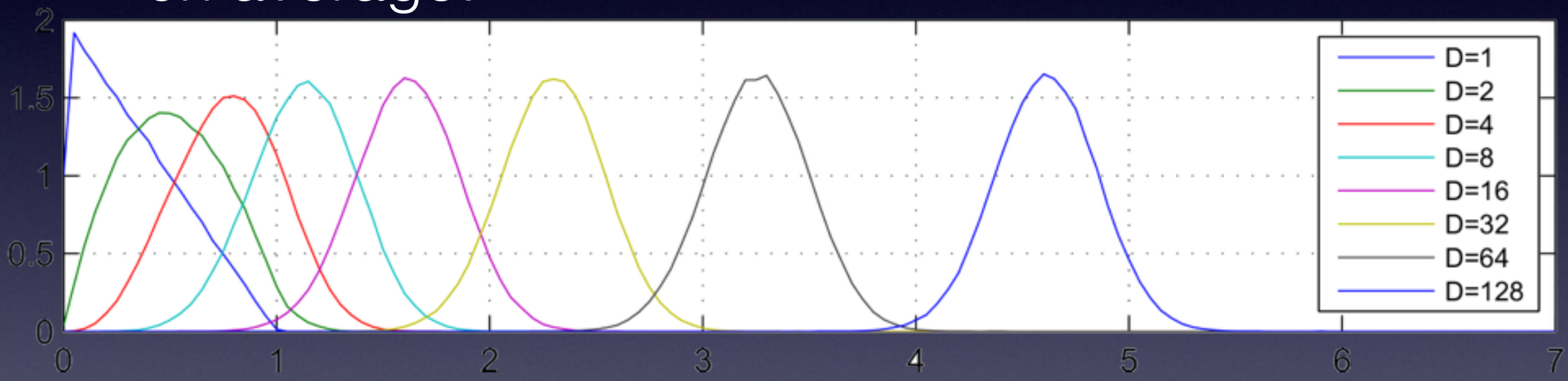
- Hashing
  Geometric Hashing (GH), Locality Sensitive Hashing (LSH)

Motivation

- Finding *the best match* to a query descriptor **q** in a database with N prototypes $\mathbf{p}_1 \ldots \mathbf{p}_N$ costs O(N).
- For a database with thousands or millions of descriptors this is expensive.
- A search tree can find *several good matches* (near neighbours) in O(log N) time.
- A hash table can find *a good match* in O(1) time.

High dimensional spaces

- Distances in high dimensional spaces are higher on average!



Expected distance for two points in D-dim unit cube

- Small distances are unlikely in $\mathbb{R}^D$ for high D.

High dimensional spaces

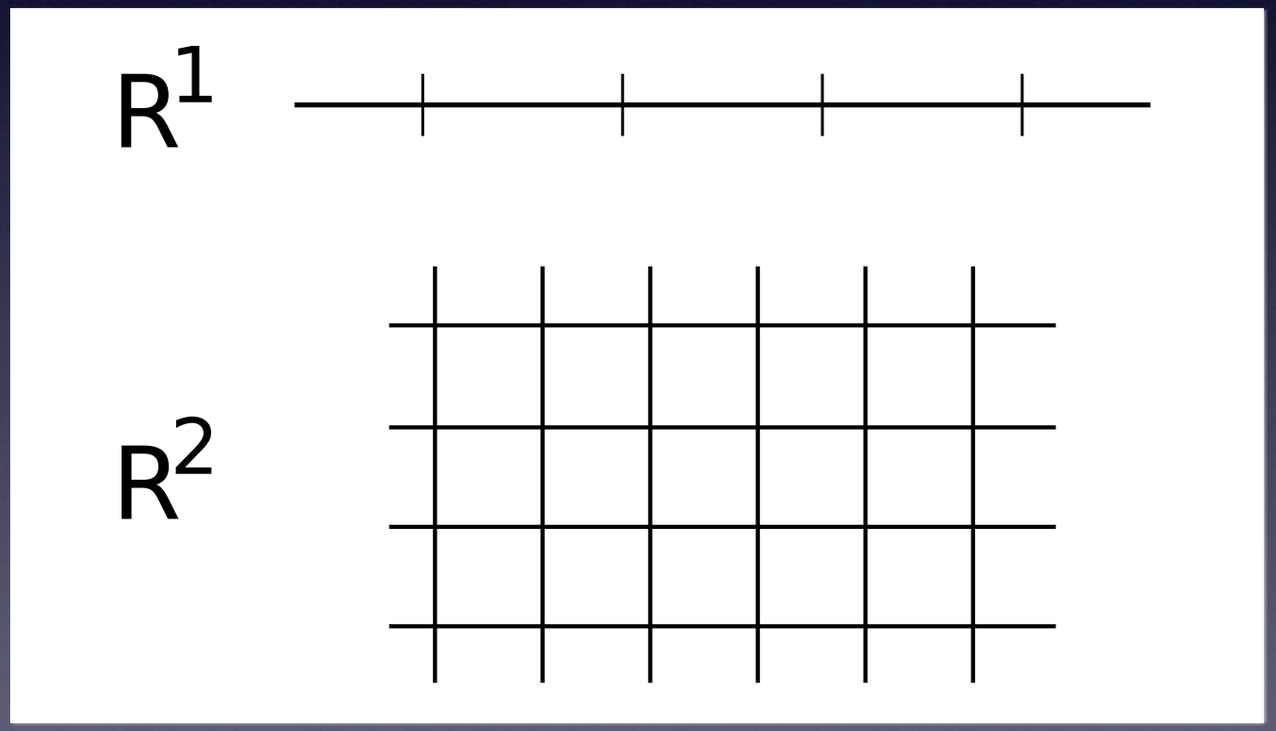
- Volume shrinks relative to area.
  Example: unit “ball”

dimension	volume	area	volume/area
1	$2r$	2	r
2	πr^2	$2\pi r$	$r/2$
3	$4\pi r^3/3$	$4\pi r^2$	$r/3$
4	$\pi^2 r^4/2$	$2\pi^2 r^3$	$r/4$

- This means that a decision region in $\mathbb{R}^D$ has increasingly more edges as D increases.

High dimensional spaces

- E.g. box decision regions have 2 edges in R^1, 4 in R^2, 6 in R^3, 8 in R^4,... 2D in R^D



...

Binary search

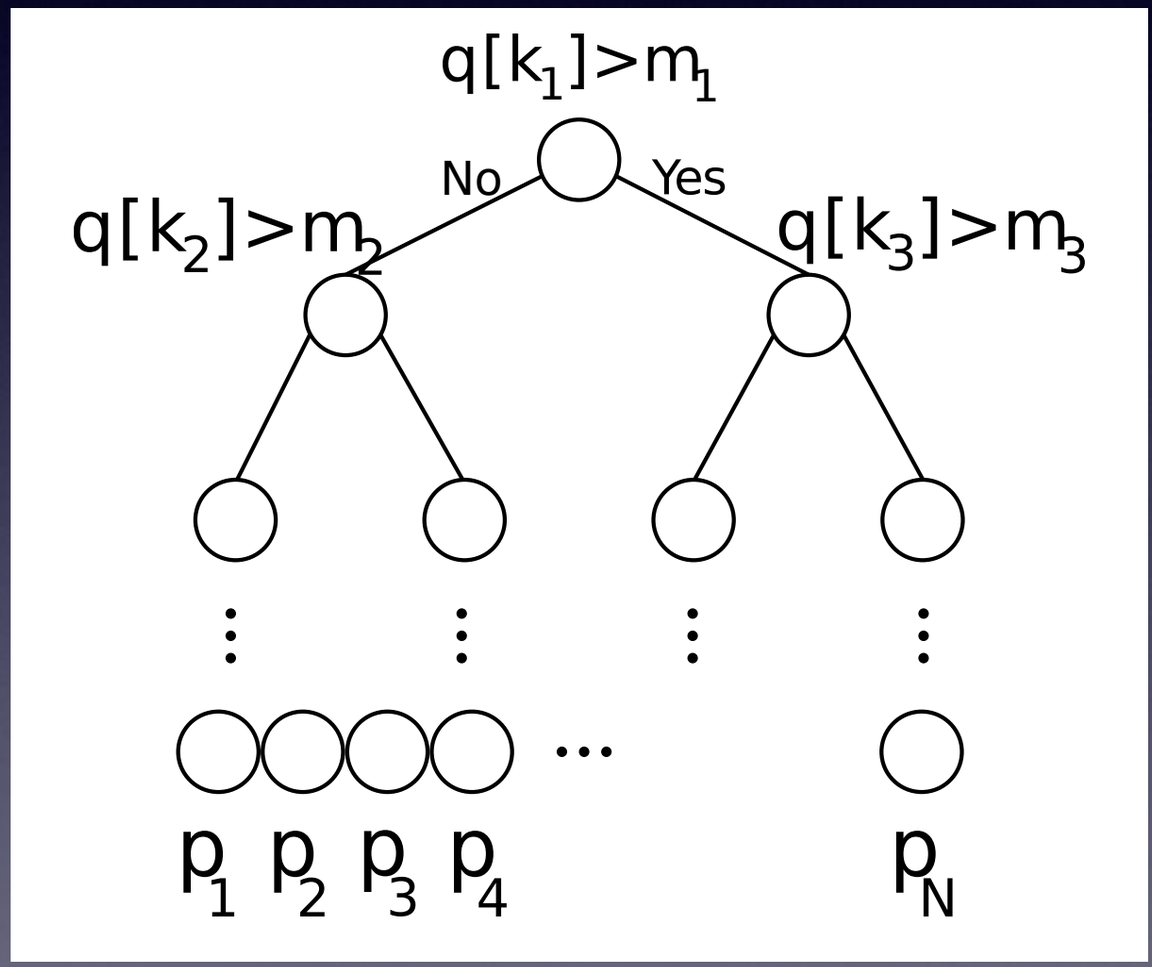
- Binary search to find scalar q in a list with N entries $p_1 \ldots p_N$:

1. Sort values such that $p_n \geq p_{n+1} \quad \forall n$
2. Set l=1 h=N
3. while
4. $m = \lfloor (l+h)/2 \rfloor$
5. if m==l break
6. if q>p(m) l=m
7. else h=m

- Complexity O(log(N)), exact solution found.

kD-Trees

- Generalization of binary search to nD:



kD-Trees

- Binary search only works in 1D,
  in higher dimensions the kD-tree gives a
  near neighbour.

- Tree construction algorithm:

  1. Select dimension k_n with largest variance
  2. Split dataset in two along selected
     dimension at median value, m_n.
  3. Repeat for each of the subsets.

kD-Trees

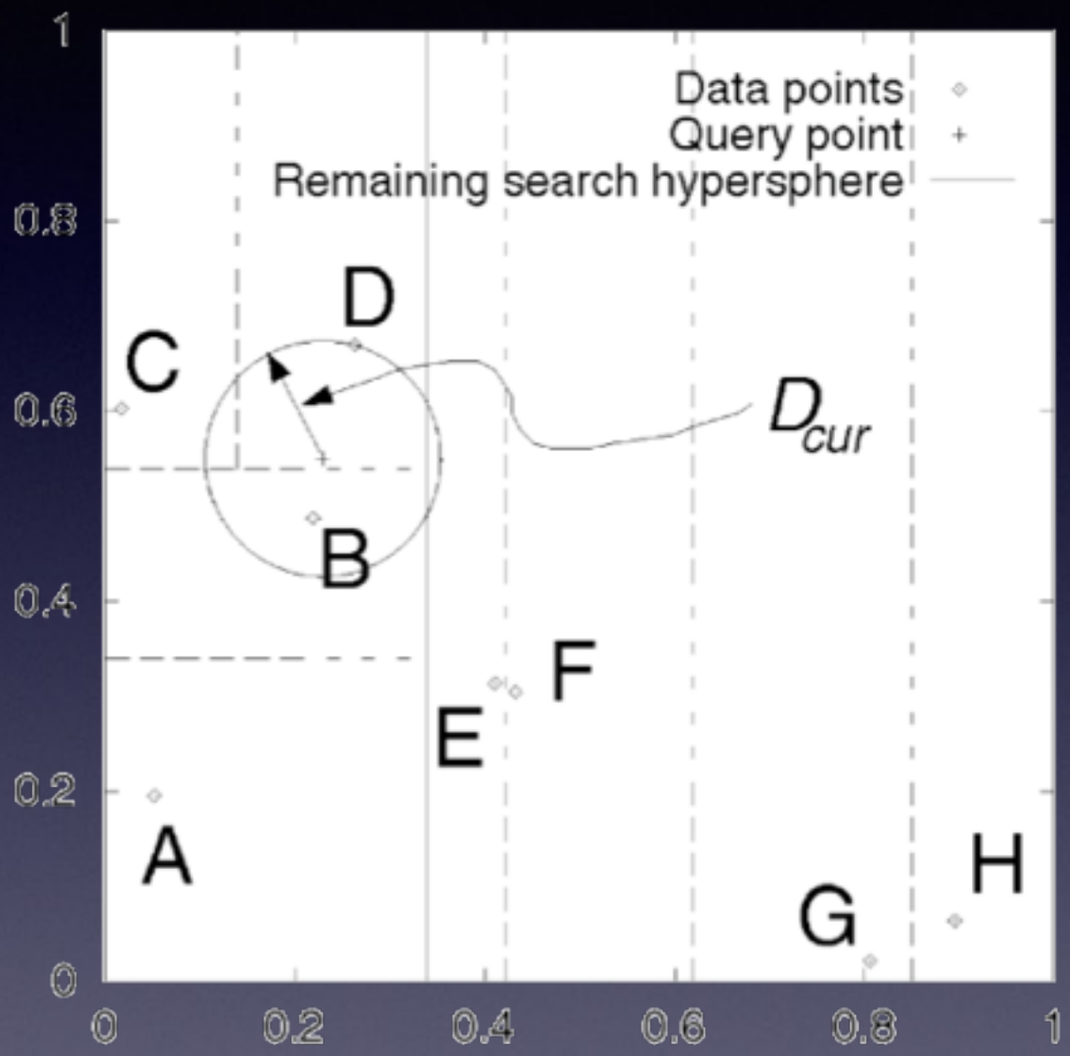
- Search for one neighbour is just one pass down the tree, and thus computation time is proportional to tree depth, d
- Tree depth $d = \lceil \log_2 N \rceil$
- To find more neighbours, the original algorithm suggested a depth-first search with branch pruning.
- If $e_{curr} < q[k_n] - m_n$ then skip branch.

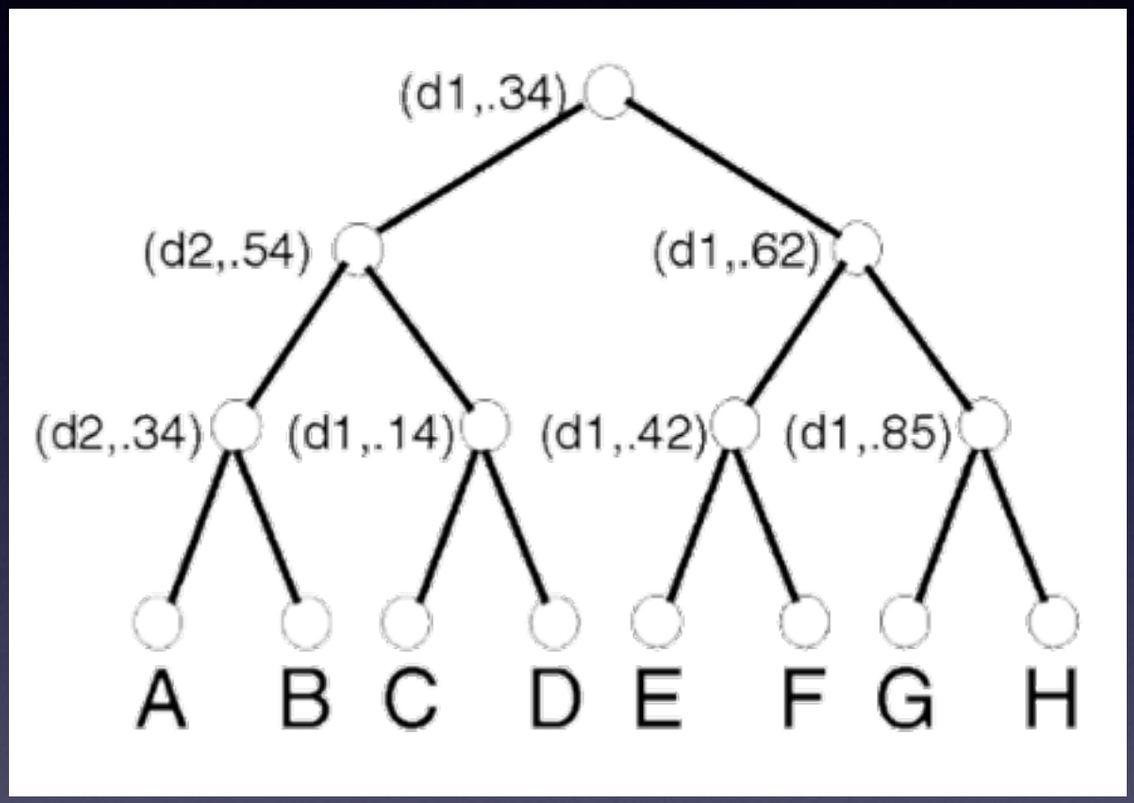
Best-bin-first

- Depth-first search works poorly in high dimensional spaces, and thus [Beis&Lowe CVPR'97] suggest a ***best-first search*** instead.
- Algorithm:

1. At each node, store the distance $e_n=q[k_n]-m_n$ in a *priority queue*. Always insert lowest value first.
2. Go down alternate branch of the first node in the queue if $e_n<e_{curr}$

Best-bin-first





Multiple randomized kD-trees

- Multiple randomized kD-trees [Silpa-Anan& Hartley, CVPR'08]

- Create multiple kD-trees with small random variations:
  In tree construction, select the D dimensions with largest variance, draw one at random (e.g. D=5).

- Back-track in all trees in parallel, using best-first search.

Ball Trees

- [Omohundro TR'89], *Metric Tree* [Uhlmann IPL'91]
- A ***radius nearest neighbour*** (RNN) method: find all neighbours within a distance ϵ from the query vector
- Each node in tree has a centre **p**, and a radius r

  p is average of all leaves
  r is maximum distance from p to a leaf

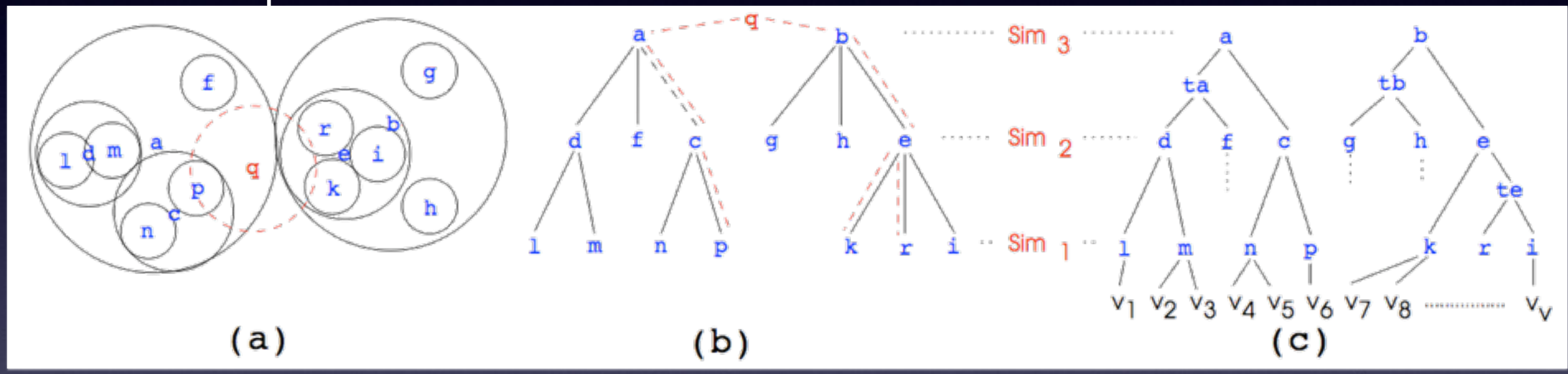
Ball Trees

- An optimal ball tree is constructed bottom up. Very expensive. E.g. using *agglomerative clustering*:
  1. Set each sample to be one cluster
  2. Merge the two most similar clusters
  3. Repeat step 2 until no clusters are left.

- Agglomerative clustering generates a *dendrogram*, or similarity tree. This can be pruned using various heuristics to form the ball tree.

Ball Trees

- Example of a search:



Leibe&Mikolajczyk&Schiele, BMVC'06

- At each node, the distances to circle centres are computed, and compared to the radius.

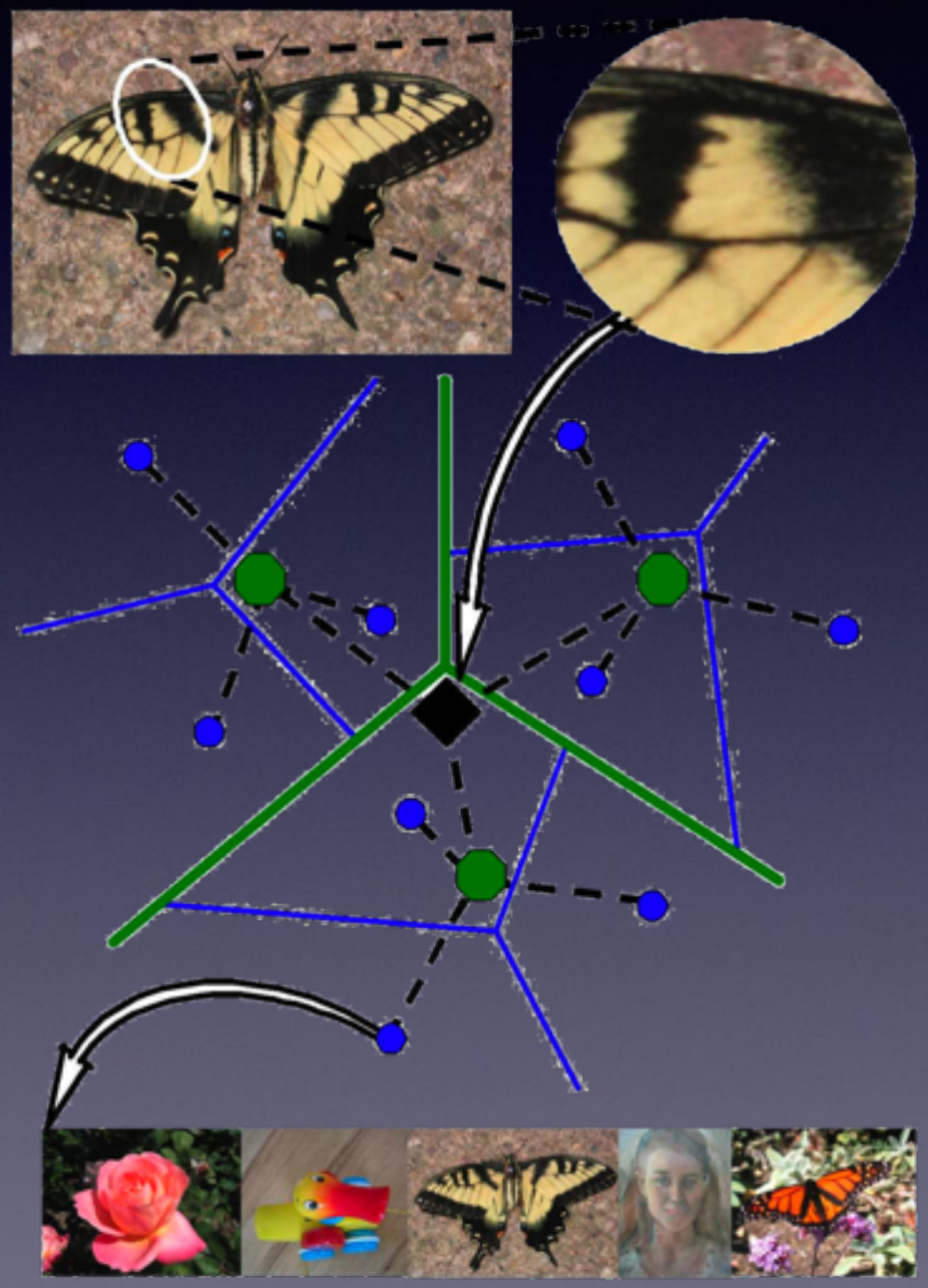
Ball Trees

- Advantage: Good if RNN is needed.
  I.e. find all neighbours with $d<d_{max}$

- Disadvantages:
  - Tree construction algorithm does not scale to very large datasets
  - A ball in $\mathbb{R}^D$ is not such a useful region shape if sample density varies in the feature space.

K-means Tree

- E.g. David Nistér and Henrik Stewénius, *Scalable Recognition with a Vocabulary Tree*, CVPR06



- Hierarchical modification of the visual words idea from LE5

K-means Tree

- Building the tree:

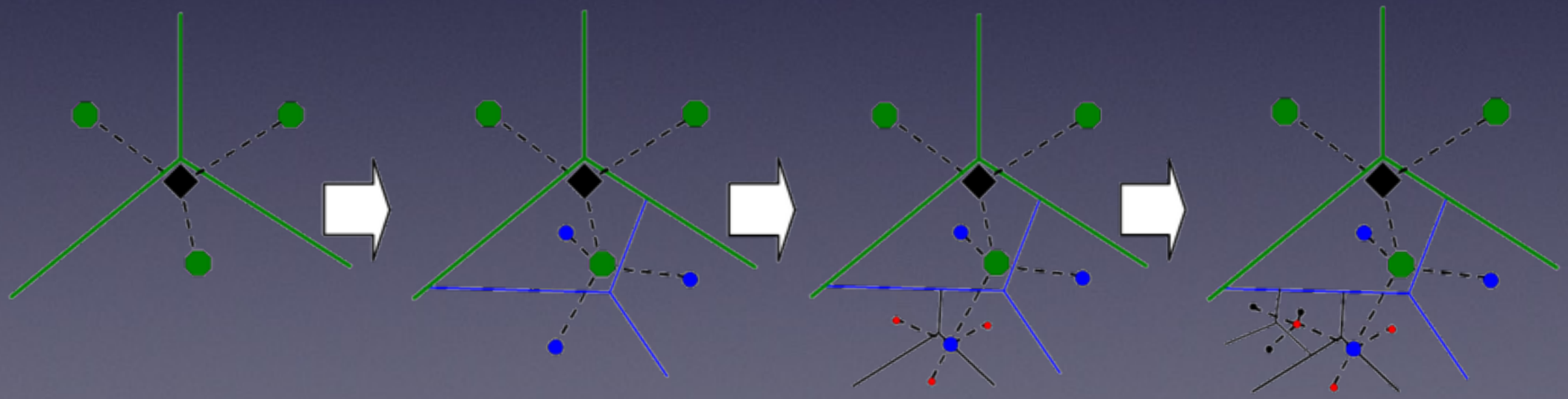
    1. Run K-means with e.g. K=10 on whole dataset.
    2. Partition dataset into K subsets using Voronoi regions
    3. Apply algorithm recursively on subsets.

- The tree gets branching factor K.

K-means Tree

- Using the tree:

    1. Compare query vector to prototypes at current level.
    2. Go down best branch

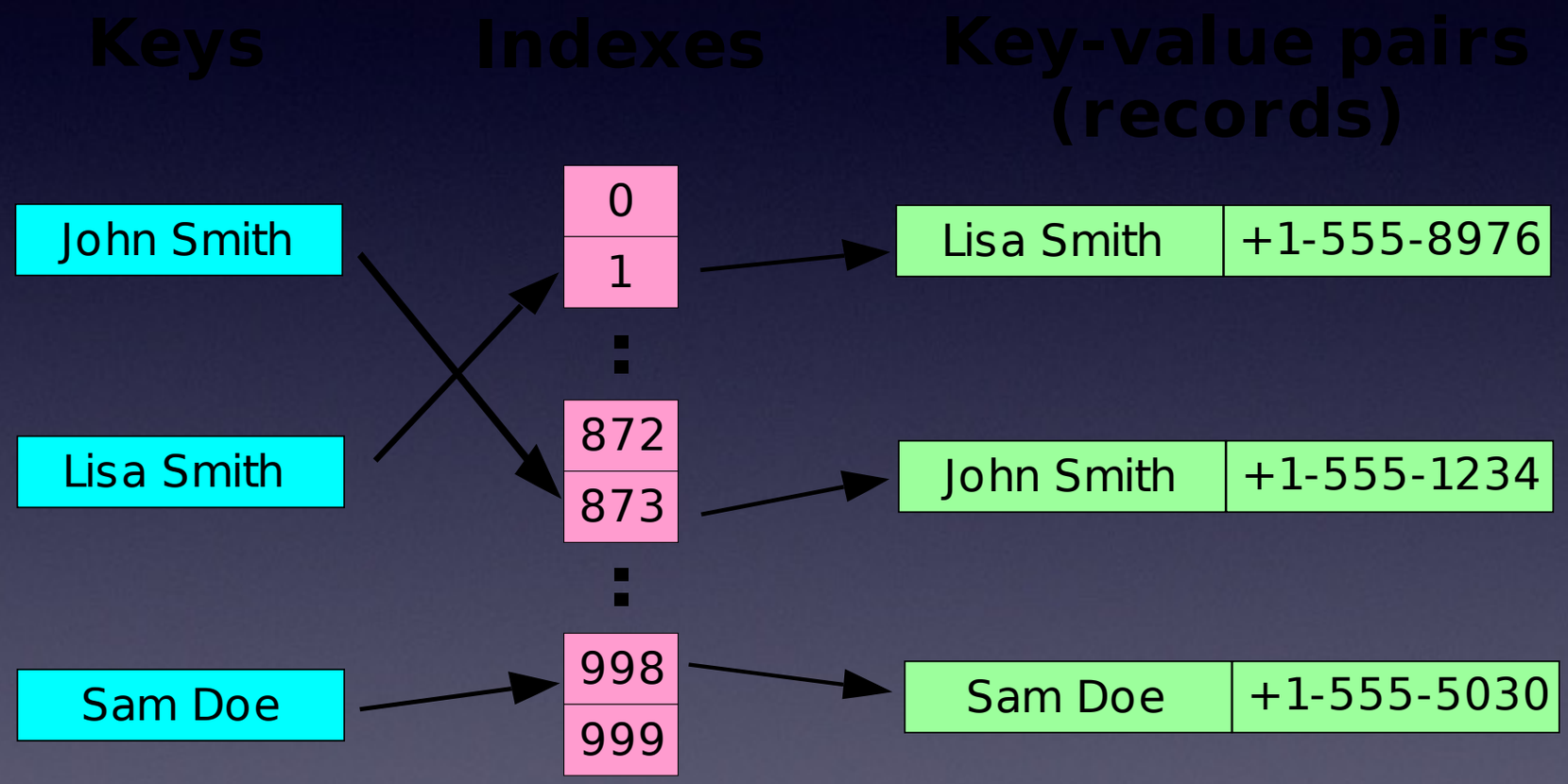


K-means Tree

- Used to compute a TF-IDF bag-of-words vector quickly.
- Much faster than non-hierarchical visual words algorithm.
- As in the kD-tree, the terminal leaf node is a near neighbour.
- The best-bin-first strategy makes the K-means tree a strong contender for ANN (today's paper).

Hash Tables

- An efficient way to perform lookup.



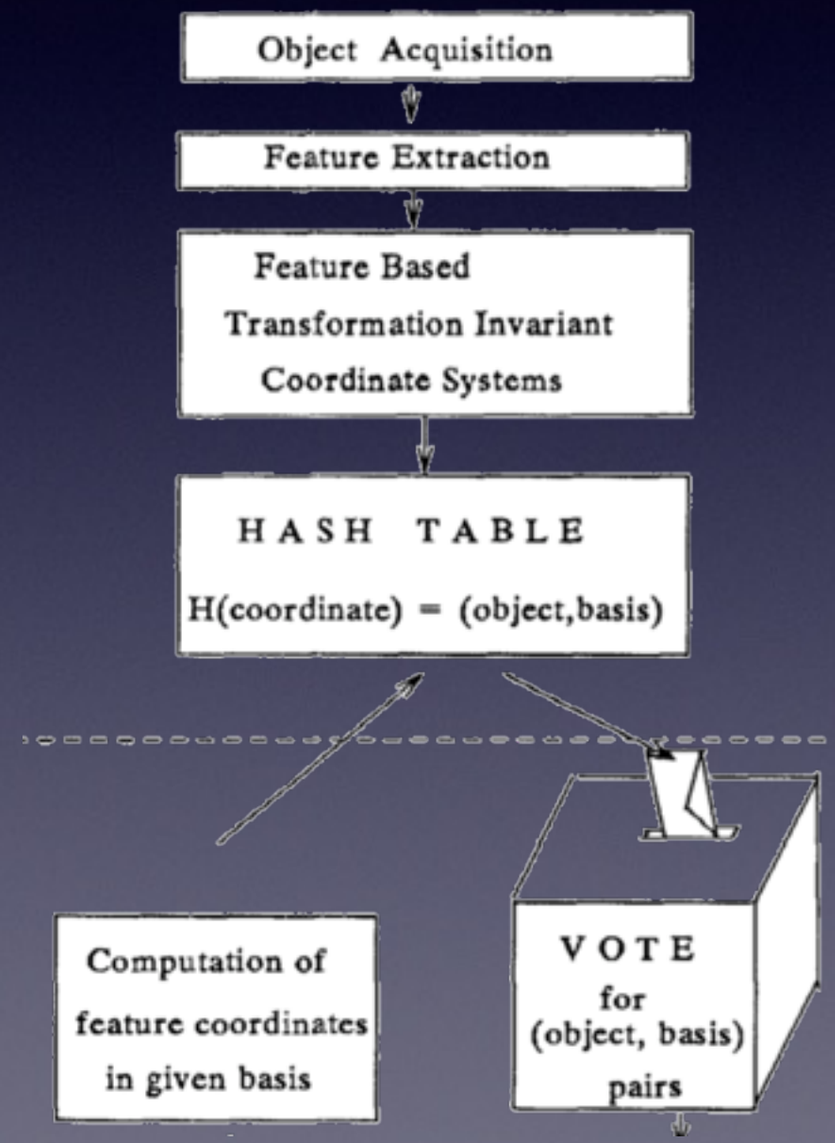
- Each key is converted to an index using a *hashing function*: index=H(key)

Hash Tables

- Lookup is O(1) instead of e.g. O(N) in a list, O(log N) in a sorted list/tree etc.
- Collisions can happen. i.e. different keys get the same index. Solved e.g. using *chaining* (linked lists), or *linear probing* (insertion at next free slot).
- Linear probing typically wants a <80% filled table.
- Hashing has poor cache locality.

Geometric Hashing

- Introduced in Lamdan&Wolfson ICCV'88

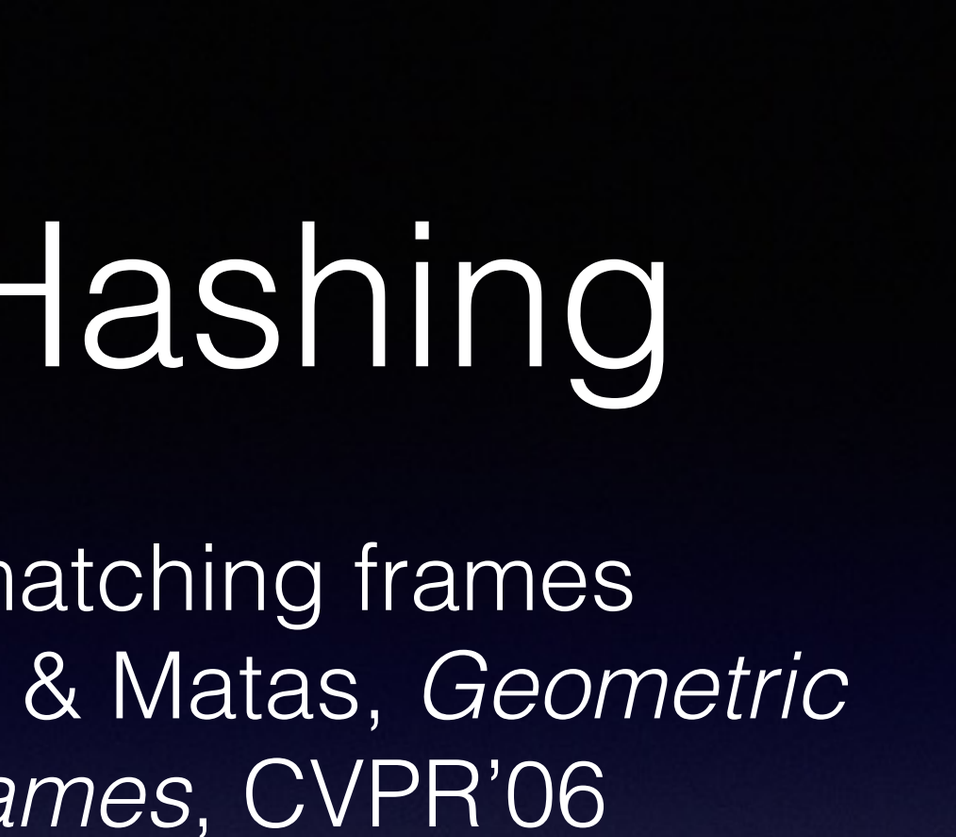


Geometric Hashing

- Modern example: Used for matching frames *without descriptors* by Chum & Matas, *Geometric Hashing with Local Affine Frames*, CVPR'06

- Use pairs of affine frames. Express frame 2 in frame 1. 25 bins for angle 16 for d_1, 6 for d_2 & d_3

- $9*10^6$ unique values for key to hash.



Geometric Hashing

- Design of H(key) is not discussed further.
- GH suffers from the same basic problem as trees: Neighbouring bins might contain the closest match. A low dimensional space is needed.
- To further deal with the neighbour problem, Chum&Matas construct 6 different tables (for 6 different frame constructions) and run them in parallel.

Locality Sensitive Hashing

- LSH Introduces locality concept in general for hashing. Introduced by Indyk&Mowani at STC'98. See also MP-LSH paper [Lv et al., ICVLDB'07 2008]

- Hash functions are designed to increase risk of collision for similar data points=***be locality sensitive***.

Locality Sensitive Hashing

- Common choice: $H(\mathbf{q}) = (H_1(\mathbf{q}), \ldots, H_L(\mathbf{q}))$
- Where $H_l(\mathbf{q}) = \lfloor \hat{\mathbf{n}}_l^T \mathbf{q}/w \rfloor$
  are random quantized projections.

Locality Sensitive Hashing

- Common choice: $H(\mathbf{q}) = (H_1(\mathbf{q}), \ldots, H_L(\mathbf{q}))$
- Where $H_l(\mathbf{q}) = \lfloor \hat{\mathbf{n}}_l^T \mathbf{q}/w \rfloor$
  are random quantized projections.

- LSH probing:
  Alt 1: Construct several hash tables (randomly),
  and index them all using **q**.
  Alt 2: Construct several queries, by adding
  random noise to **q**.

Locality Sensitive Hashing

- LSH extensions:
  - Multi-probe LSH [Lv et al., ICVLDB'07]
    Instead perturbs H(**q**)
    Neighbours to H_l(**q**) are H_l(**q**)-1 and H_l(**q**)+1
    Try promising bins in sequence.
  - LSH Forest [Bawa et al ICWWW'05]
    Orders hash functions in a tree. Easier to update as new data is added. Also enables self tuning where parameters are adjusted.

Summary

- Decision regions in nD are dominated by edges
- Approximate KNN and RNN are implemented using either **search trees** or **hashes**.
- Trees are O(log(N)) and more space efficient than hashes.
- Hashes can be O(1), but most adjustments make them depend on N in practise.

Projects

- Course is 8hp:
  5hp for lectures+articles+exam
  3hp for project.
- Project part is 2 weeks of:
  programming&research
  writing a small report (a conference submission will also do).

Project examples

1. Repeat the comparison of LSH and FLANN in today's paper on your own binary feature set (test speedup vs. precision). (or with Forest LSH instead of MP-LSH).

2. Use e.g. VLfeat (http://www.vlfeat.org) to implement a bag-of-words recognition system. Test how system parameters affect result.

3. Integrate SFOP detector with BRIEF descriptor and compare results to e.g. SIFT.

Project examples

4. Implement and test a spatial verification for BoF style recognition (e.g. using similarity, or affine transform) of feature locations.

5. Learn a matching metric, and compare it to least squares matching.

6. Compare Chi^2, EMD and least-squares on a problem of choice.

7. Your own suggestion.

Exam

Everyone should bring calendar for the next seminar, so we can decide on a date for the written exam.

Plan: Middle of or end of April.

Discussion

- Questions/comments today's paper:

  M. Muja and D.G. Lowe, "Scalable Nearest Neighbour Algorithms for High Dimensional Data", TPAMI 2014

Paper for next week

- Paper for next week will be announced over email later…