Reconnaissance d’objets et vision artificielle 2010

Instance-level recognition II.

Josef Sivic

http://www.di.ens.fr/~josef

INRIA, WILLOW, ENS/INRIA/CNRS UMR 8548

Laboratoire d’Informatique, Ecole Normale Supérieure, Paris

With slides from: O. Chum, K. Grauman, S. Lazebnik, B. Leibe, D. Lowe, J. Philbin, J. Ponce, D. Nister, C. Schmid, N. Snavely, A. Zisserman

Outline – the rest of the lecture

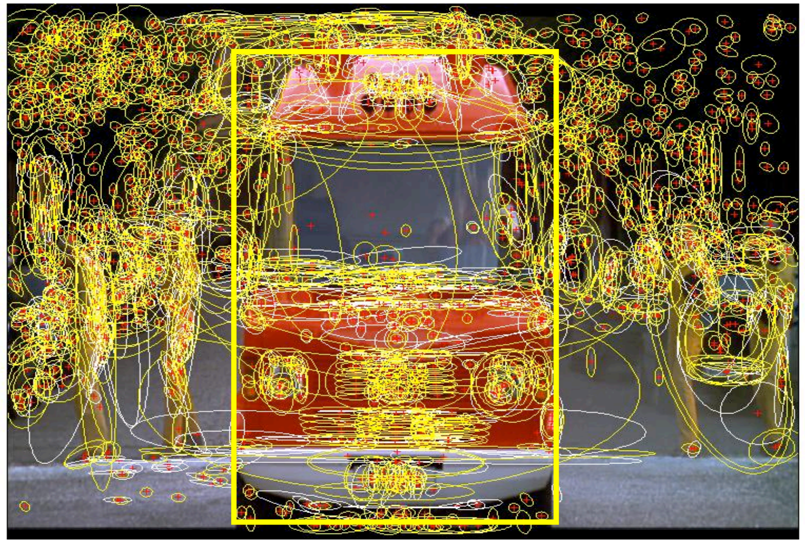
Part 1. Matching and recognition with local features

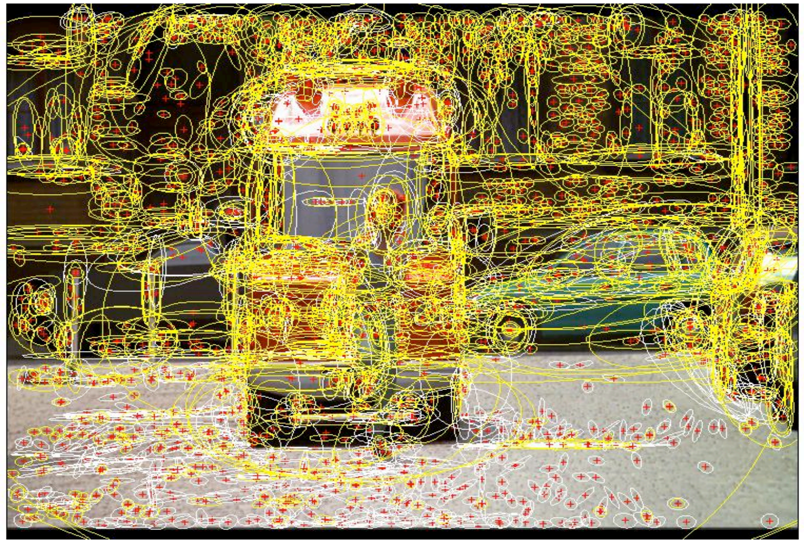
- Correspondence
- Semi-local and global geometric relations
- Robust estimation – RANSAC and Hough Transform

Part 2. Going large-scale

- **Approximate nearest neighbour matching**
- **Bag-of-visual-words representation**
- **Efficient visual search and extensions**
- **Applications**

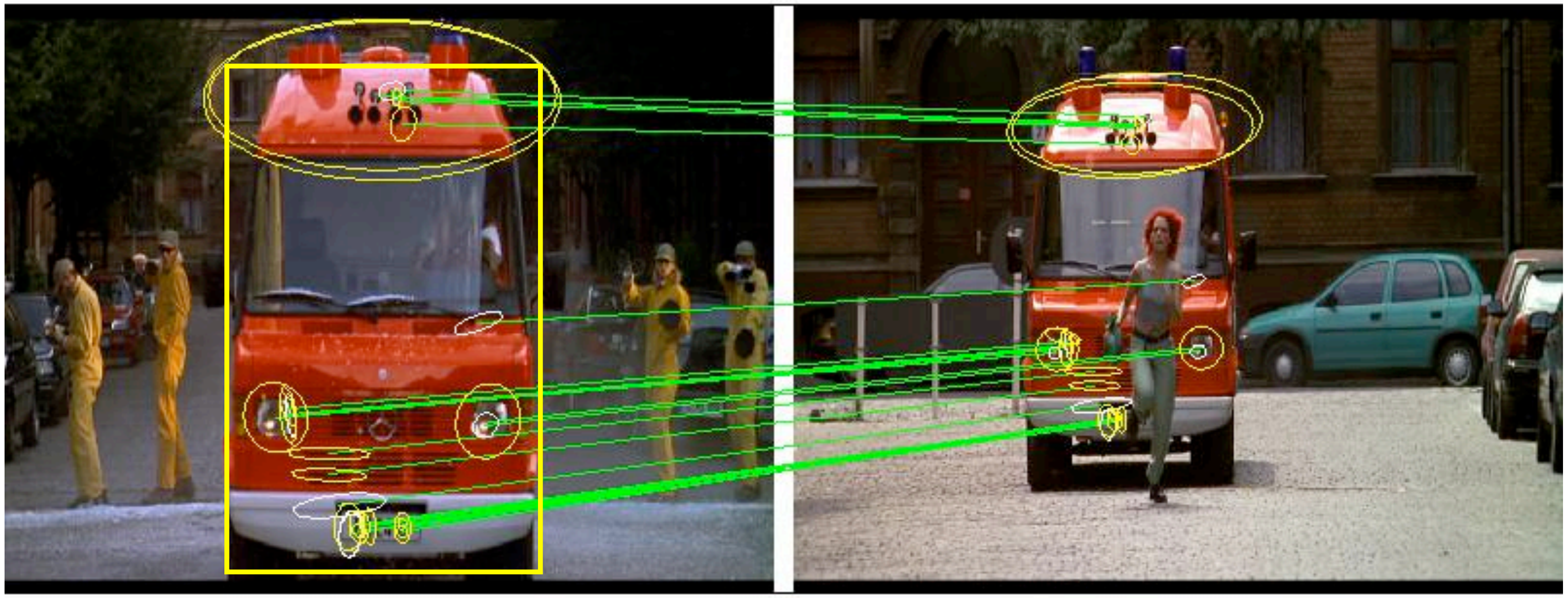
Example II: Two images again





1000+ descriptors per image

Match regions between frames using SIFT descriptors and spatial consistency



Multiple regions overcome problem of partial occlusion

Approach - review

1. Establish tentative (or putative) correspondence based on local appearance of individual features (now)

2. Verify matches based on semi-local / global geometric relations (You have just seen this).

What about multiple images?

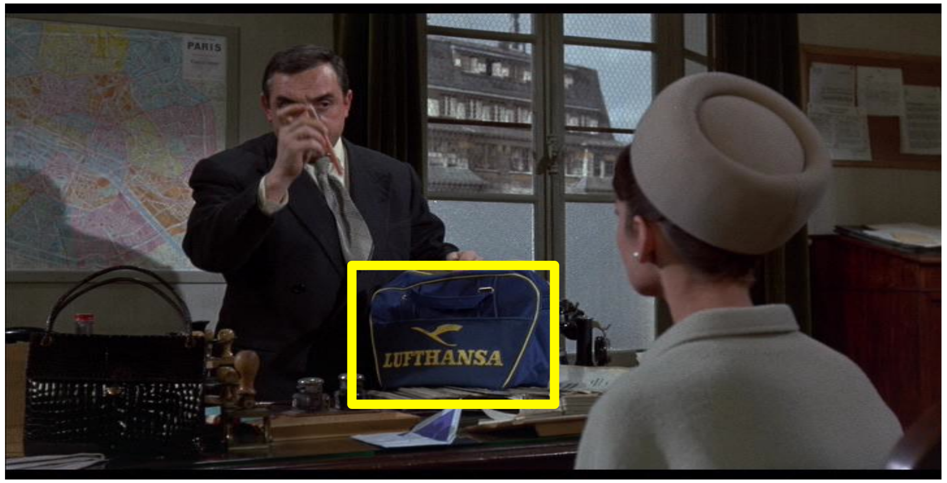
- So far, we have seen successful matching of a query image to a single target image using local features.

- How to generalize this strategy to multiple target images with reasonable complexity?

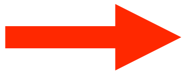
    - 10, 10^2, 10^3, …, $\mathbf{10^7}$, … 10^{10} images?

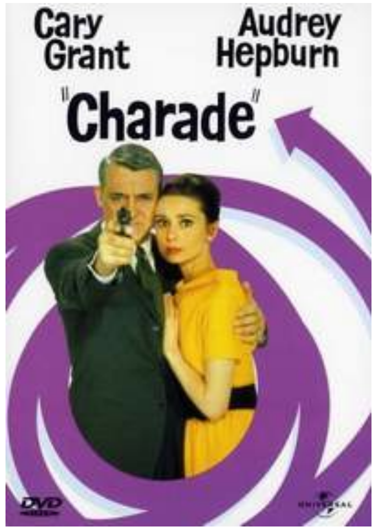
Example: Visual search in an entire feature length movie

Visually defined query



"Find this bag"





"Charade" [Donen, 1963]

Demo:
http://www.robots.ox.ac.uk/~vgg/research/vgoogle/index.html

History of “large scale” visual search with local regions

Schmid and Mohr ’97	– 1k images
Sivic and Zisserman’03	– 5k images
Nister and Stewenius’06	– 50k images (1M)
Philbin et al.’07	– 100k images
Chum et al.’07 + Jegou et al.’07	– 1M images
Chum et al.’08	– 5M images
Jegou et al. ’09	– 10M images

All on a single machine in ~ 1 second!

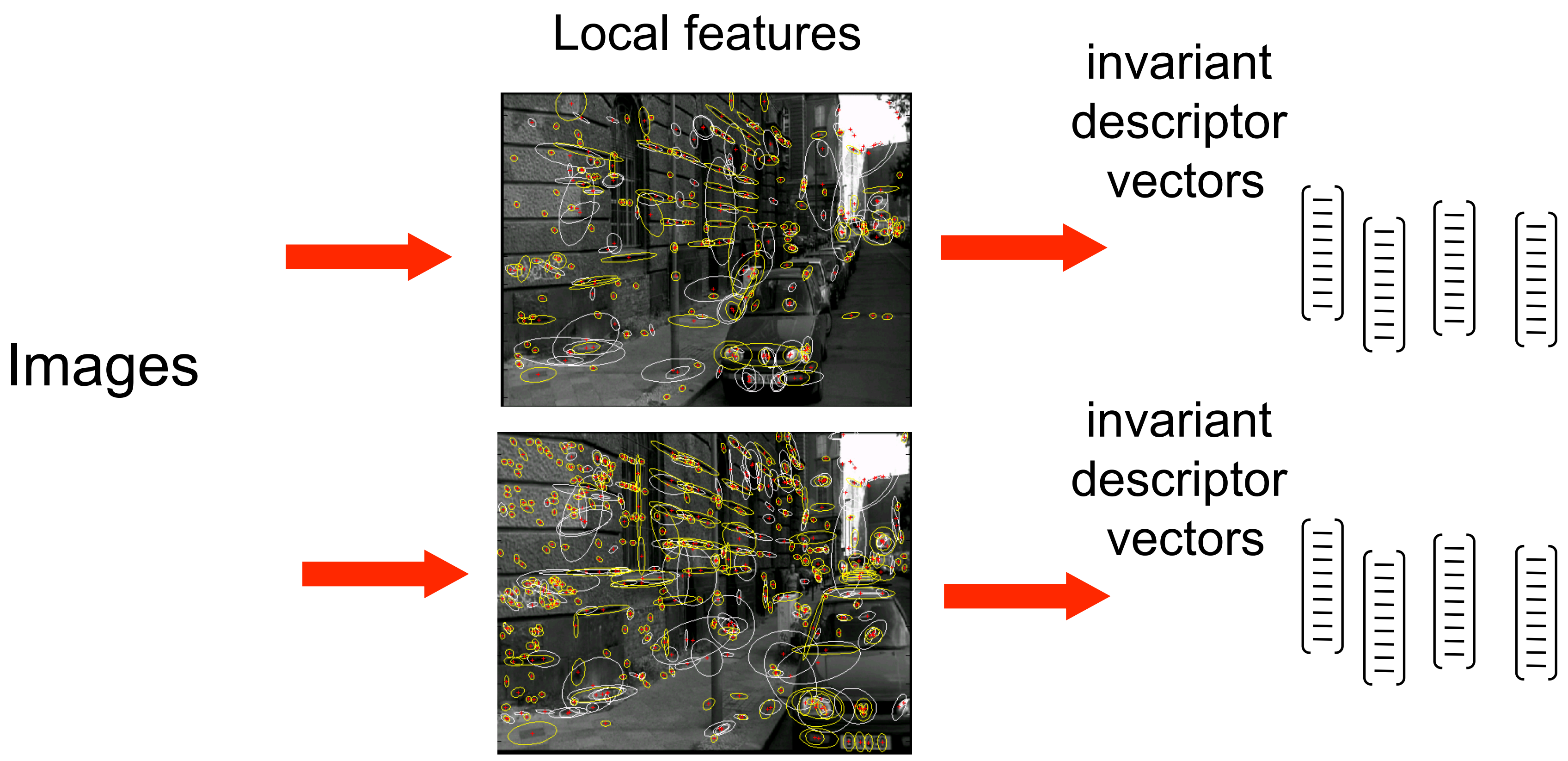
Two strategies

1. Efficient approximate nearest neighbour search on local feature descriptors.

2. Quantize descriptors into a “visual vocabulary” and use efficient techniques from text retrieval.

   (Bag-of-words representation)

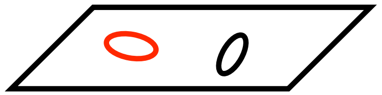
Strategy I: Efficient approximate NN search

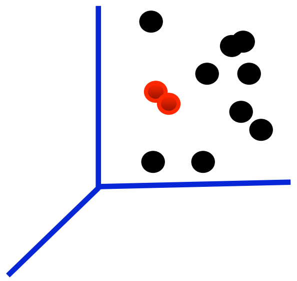


1. Compute local features in each image independently (Part 1)
2. “Label” each feature by a descriptor vector based on its intensity (Part 1)
3. Finding corresponding features is transformed to finding nearest neighbour vectors
4. Rank matched images by number of (tentatively) corresponding regions
5. Verify top ranked images based on spatial consistency (Part 2)

Finding nearest neighbour vectors

Establish correspondences between object model image and images in the database by **nearest neighbour matching** on SIFT vectors


Model image


128D descriptor space

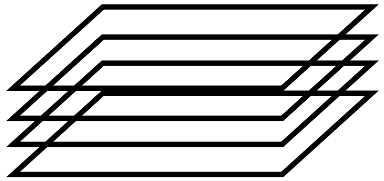


Image database

Solve following problem for all feature vectors, $\mathbf{x}_j \in \mathcal{R}^{128}$, in the query image:

$$\forall j \; NN(j) = \arg\min_i ||\mathbf{x}_i - \mathbf{x}_j||$$

where, $\mathbf{x}_i \in \mathcal{R}^{128}$, are features from all the database images.

Quick look at the complexity of the NN-search

N … images

M … regions per image (~1000)

D … dimension of the descriptor (~128)

Exhaustive linear search: O(M NMD)

Example:

- Matching two images (N=1), each having 1000 SIFT descriptors
  Nearest neighbors search: 0.4 s (2 GHz CPU, implemenation in C)
- Memory footprint: 1000 * 128 = 128kB / image

# of images	CPU time	Memory req.
N = 1,000 …	~7min	(~100MB)
N = 10,000 …	~1h7min	(~ 1GB)
…		
N = 10^7	~115 days	(~ 1TB)
…		
All images on Facebook:		
N = 10^{10} …	~300 years	(~ 1PB)

Nearest-neighbor matching

Solve following problem for all feature vectors, $\mathbf{x}_j$, in the query image:

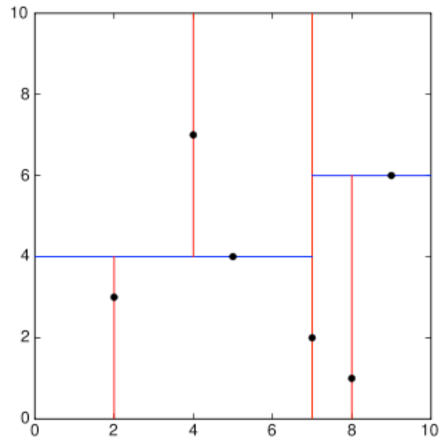
$$\forall j \; NN(j) = \arg\min_i ||\mathbf{x}_i - \mathbf{x}_j||$$

where $\mathbf{x}_i$ are features in database images.

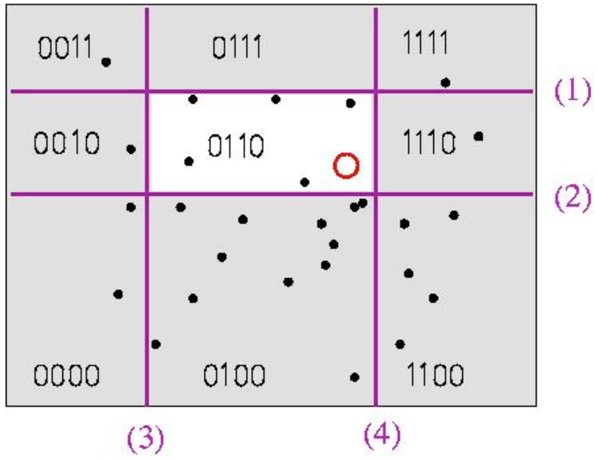
Nearest-neighbour matching is the major computational bottleneck

- Linear search performs *dn* operations for *n* features in the database and *d* dimensions
- No exact methods are faster than linear search for d>10
- Approximate methods can be much faster, but at the cost of missing some correct matches. Failure rate gets worse for large datasets.

Indexing local features: approximate nearest neighbor search



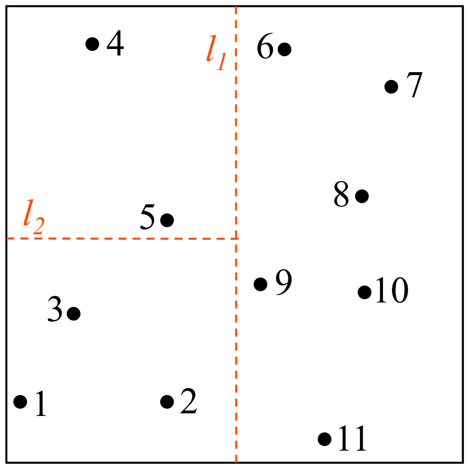
Best-Bin First (BBF), a variant of k-d trees that uses priority queue to examine most promising branches first [Beis & Lowe, CVPR 1997]

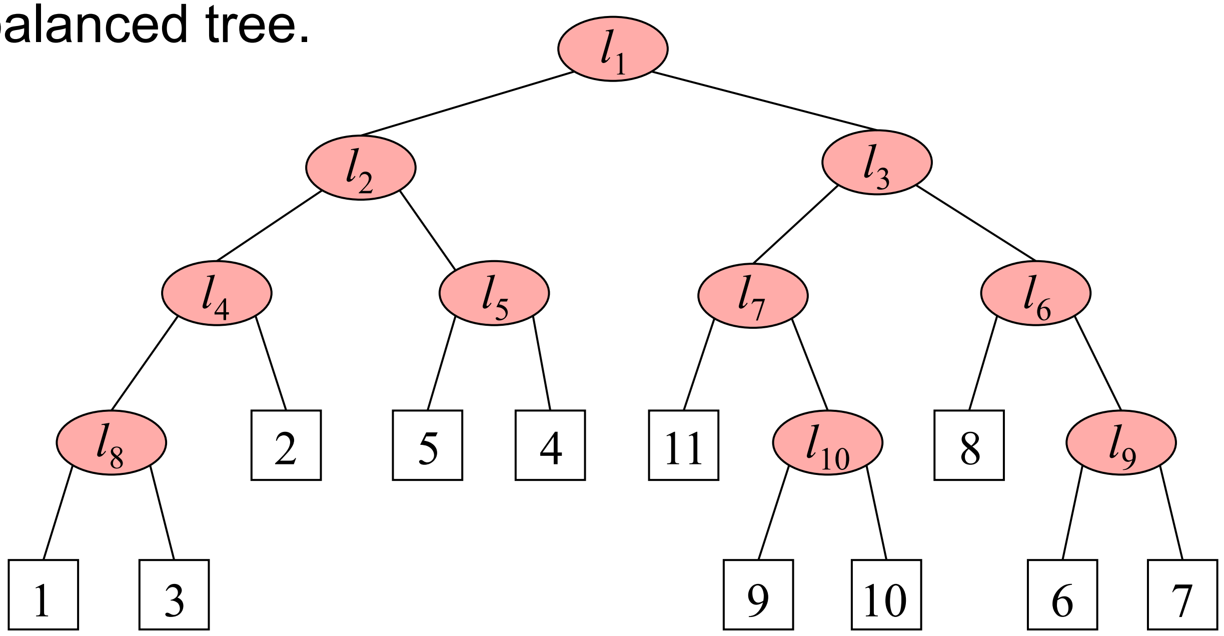


Locality-Sensitive Hashing (LSH), a randomized hashing technique using hash functions that map similar points to the same bin, with high probability [Indyk & Motwani, 1998]

K-d tree

- K-d tree is a binary tree data structure for organizing a set of points in a K-dimensional space.
- Each internal node is associated with an axis aligned hyper-plane splitting its associated points into two sub-trees.
- Dimensions with high variance are chosen first.
- Position of the splitting hyper-plane is chosen as the mean/median of the projected points – balanced tree.

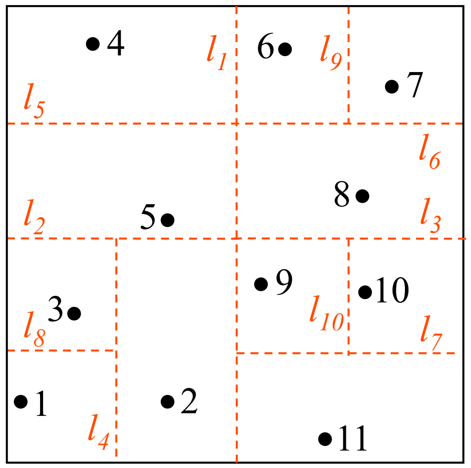


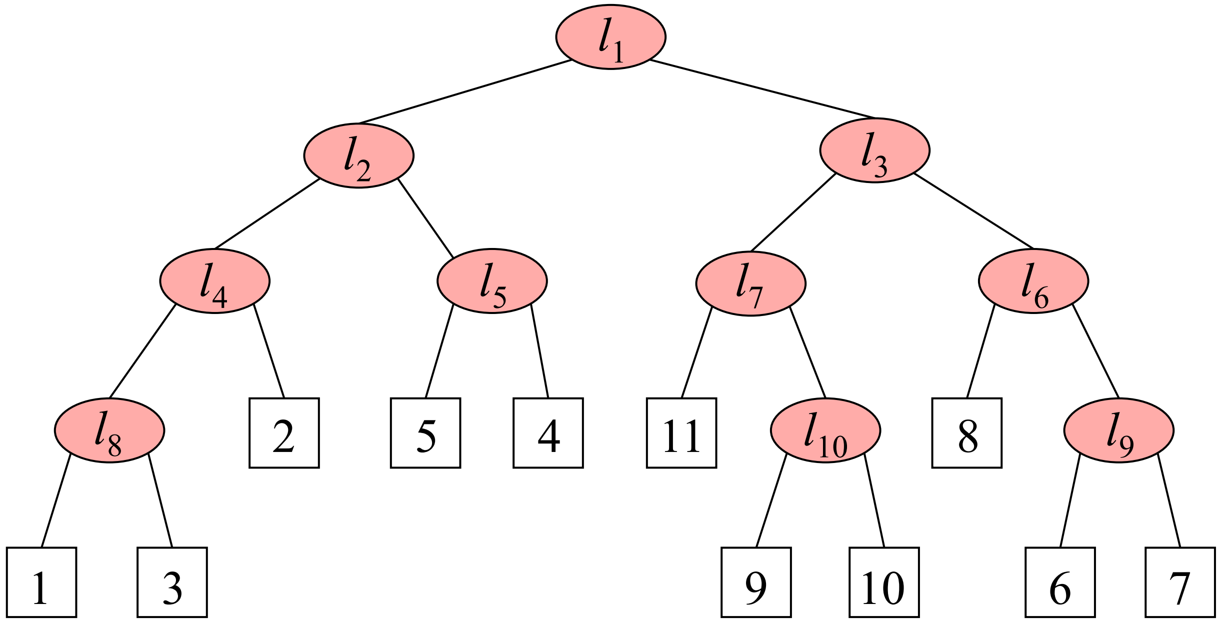


Images: Anna Atramentov

K-d tree construction

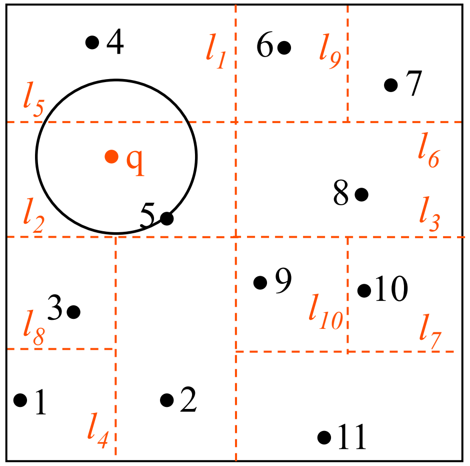
Simple 2D example

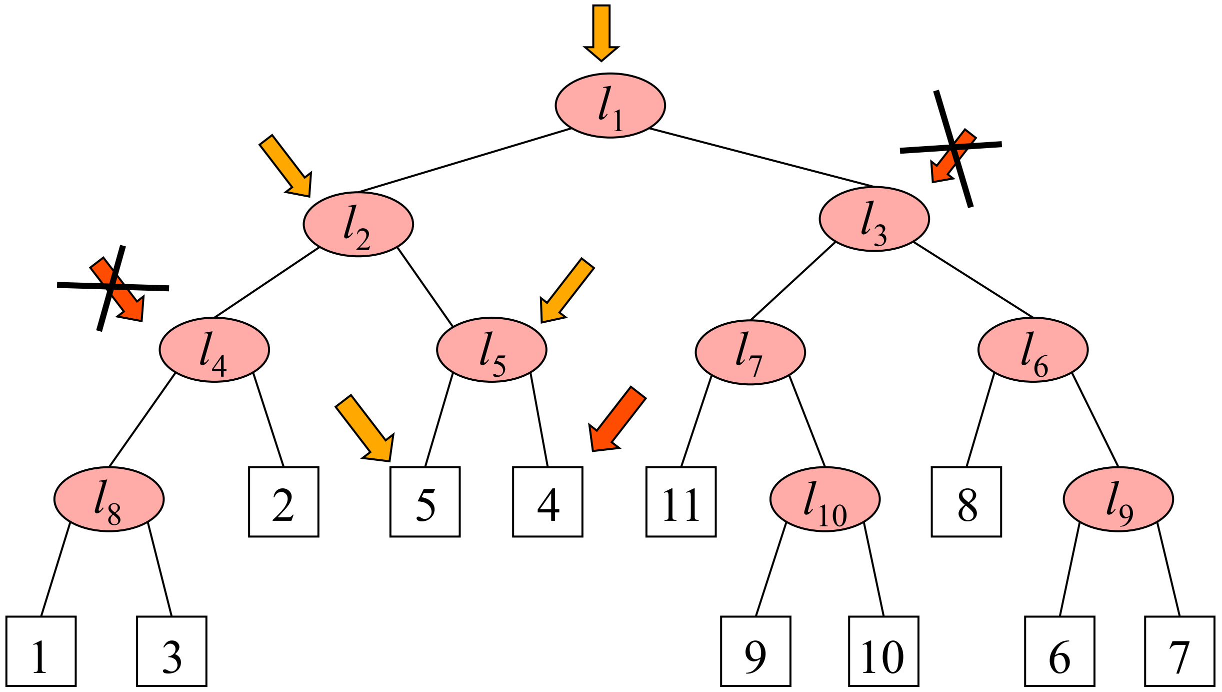




Slide credit: Anna Atramentov

K-d tree query





Slide credit: Anna Atramentov

K-d tree: Backtracking

Backtracking is necessary as the true nearest neighbor may not lie in the query cell.

But in some cases, almost all cells need to be inspected.

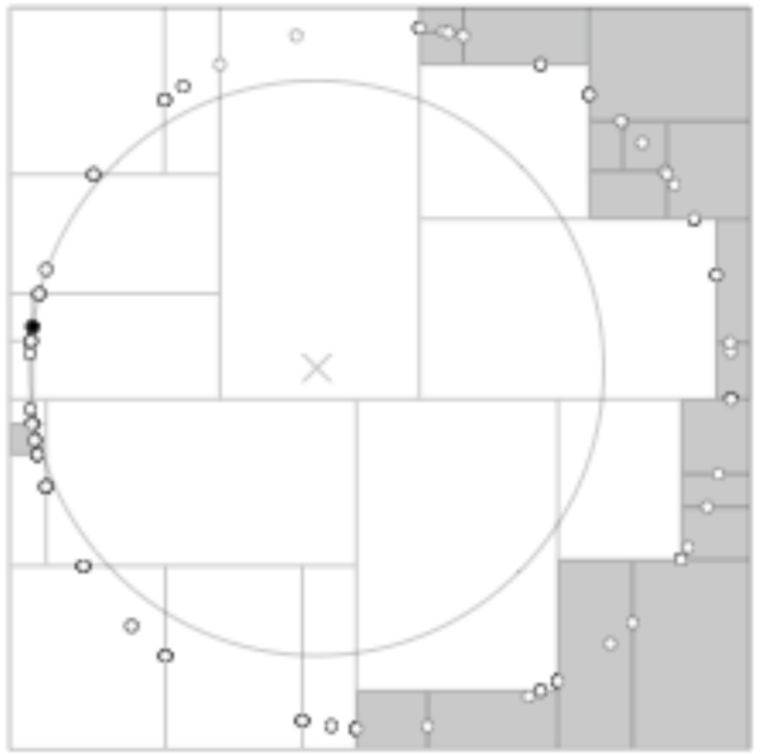


Figure 6.6

A bad distribution which forces almost all nodes to be inspected.

Figure: A. Moore

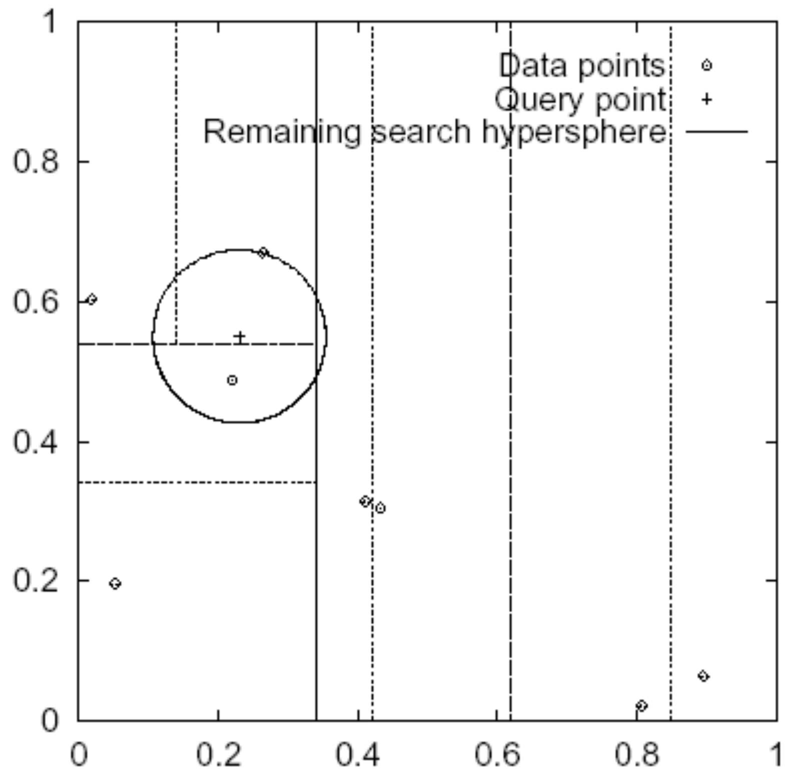
Solution: Approximate nearest neighbor K-d tree

Key ideas:

- Search k-d tree bins in order of distance from query

- Requires use of a priority queue

- Limit the number of neighbouring k-d tree bins to explore: only approximate NN is found

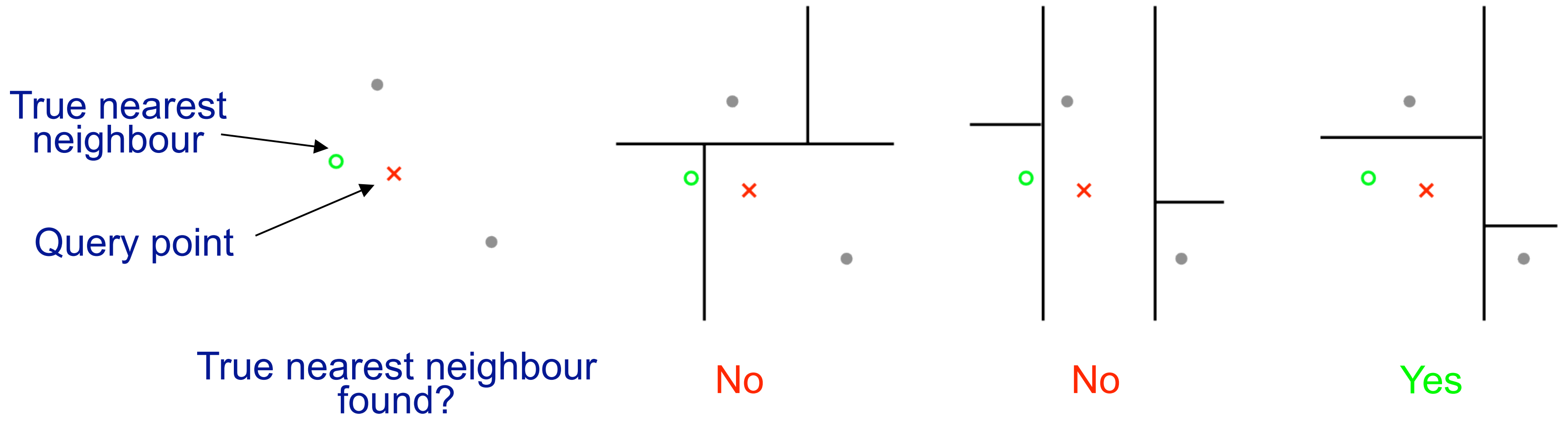


- Reduce the boundary effects by randomization

Randomized K-d trees

- How to choose the dimension to split and the splitting point?
  - Pick dimension with the highest variance
  - Split at the mean/median

- Multiple randomized trees increase the chances of finding nearby points



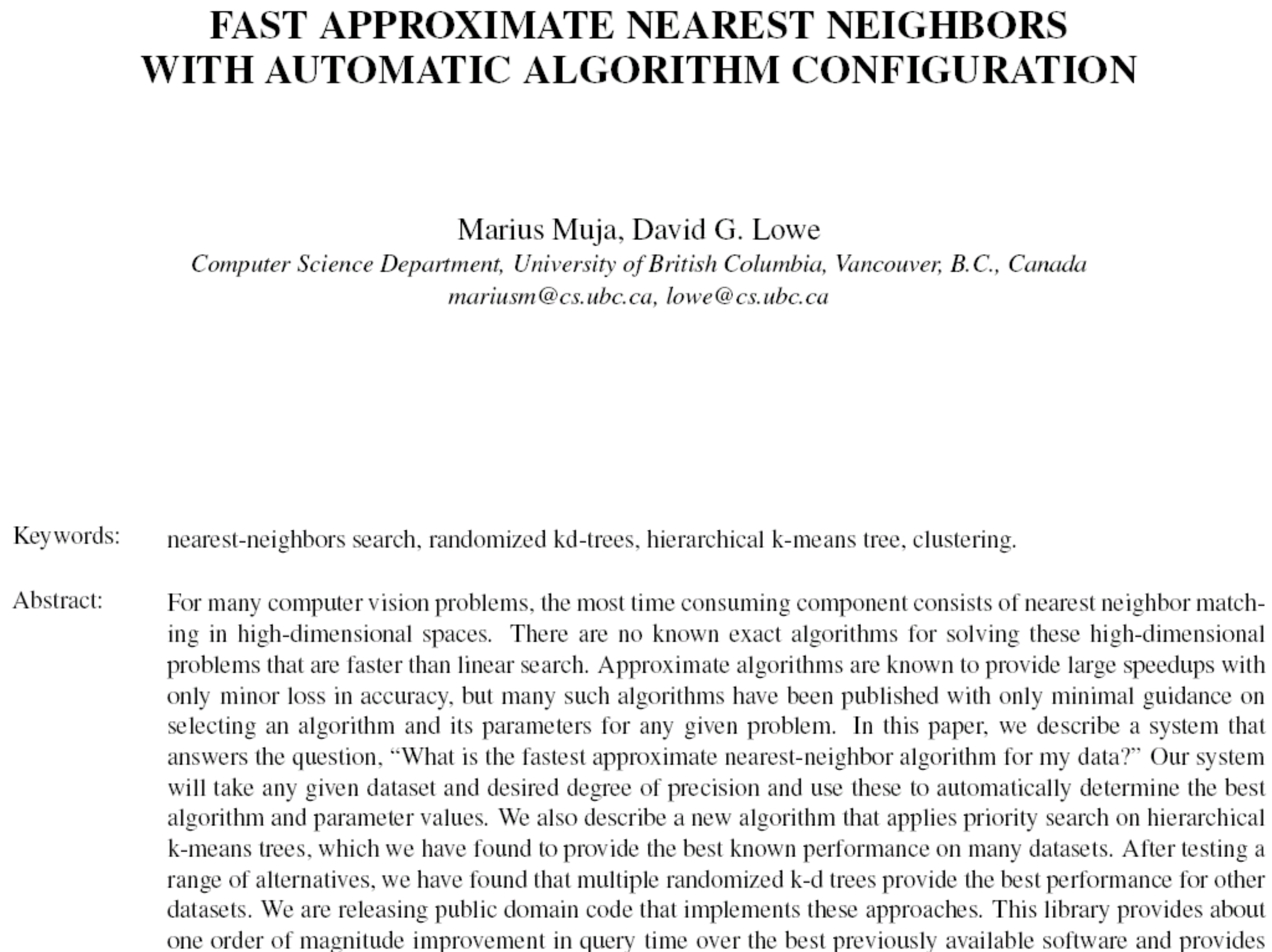
Approximate NN search using a randomized forest of K-d trees: Algorithm summary

1. Descent all (typically 8) trees to the leaf node

2. Search k-d tree bins in order of distance from query
   - Distance between the query and the bin is defined as the minimum distance between the query and any point on the bin boundary
   - Requires the use of a priority queue:
     - > During lookup an entry is added to the priority queue about the option not taken
     - > For multiple trees, the queue is shared among the trees
   - Limit the number of neighbouring K-d tree bins to explore (parameter of the algorithm, typically set to 512)

Experimental evaluation for SIFT matching

http://www.cs.ubc.ca/~lowe/papers/09muja.pdf



FAST APPROXIMATE NEAREST NEIGHBORS WITH AUTOMATIC ALGORITHM CONFIGURATION

Marius Muja, David G. Lowe

Computer Science Department, University of British Columbia, Vancouver, B.C., Canada

mariusm@cs.ubc.ca, lowe@cs.ubc.ca

Keywords: nearest-neighbors search, randomized kd-trees, hierarchical k-means tree, clustering.

Abstract: For many computer vision problems, the most time consuming component consists of nearest neighbor matching in high-dimensional spaces. There are no known exact algorithms for solving these high-dimensional problems that are faster than linear search. Approximate algorithms are known to provide large speedups with only minor loss in accuracy, but many such algorithms have been published with only minimal guidance on selecting an algorithm and its parameters for any given problem. In this paper, we describe a system that answers the question, "What is the fastest approximate nearest-neighbor algorithm for my data?" Our system will take any given dataset and desired degree of precision and use these to automatically determine the best algorithm and parameter values. We also describe a new algorithm that applies priority search on hierarchical k-means trees, which we have found to provide the best known performance on many datasets. After testing a range of alternatives, we have found that multiple randomized k-d trees provide the best performance for other datasets. We are releasing public domain code that implements these approaches. This library provides about one order of magnitude improvement in query time over the best previously available software and provides

Randomized K-d trees

Performance w.r.t. the number of trees

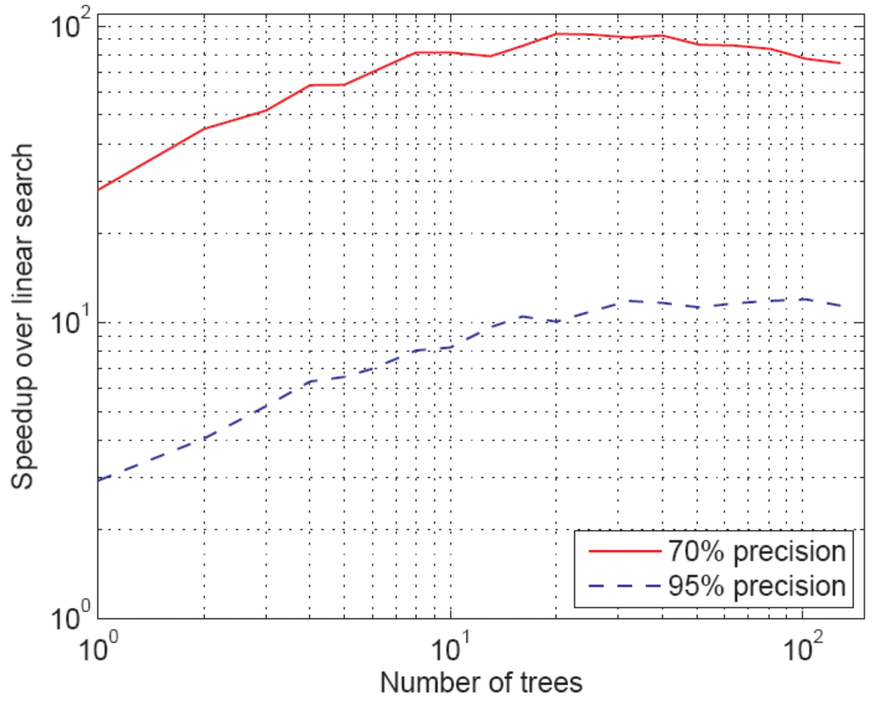


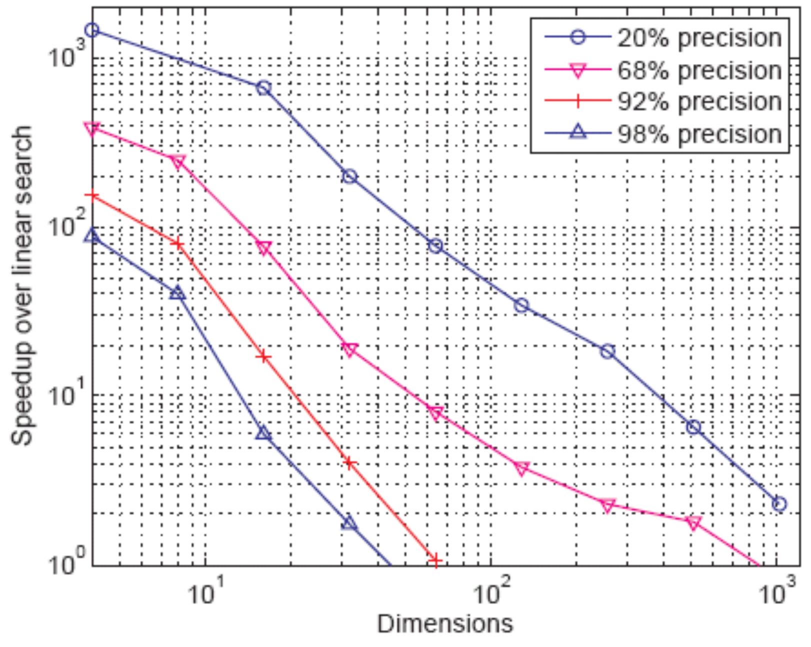
Figure 2: Speedup obtained by using multiple random kd-trees (100K SIFT features dataset)

d=128, n=100K

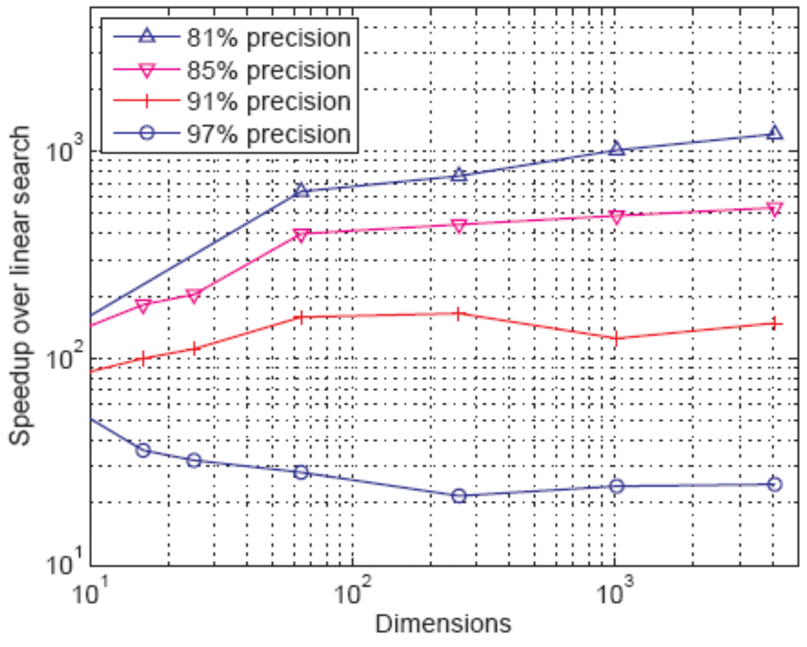
Precision: percentage of true nearest neighbours found

Randomized K-d trees

Performance w.r.t. the number of dimensions



(a) Random vectors



(b) Image patches

Figure 4: Search efficiency for data of varying dimensionality. The random vectors (a) represent the hardest case in which dimensions have no correlations, while most real-world problems behave more like the image patches (b)

Randomized K-d trees: discussion

- Find approximate nearest neighbor in O(logN) time, where N is the number of data points.
- Increased memory requirements: needs to store multiple (~8) trees
- Good performance in practice for recognition problems (NN-search for SIFT descriptors and image patches).
- Code available online:
  http://people.cs.ubc.ca/~mariusm/index.php/FLANN/FLANN

Variation: K-means tree [Muja&Lowe, 2009]

- Partition of the space is determined by recursive application of k-means clustering.
- Cell boundaries are not axis aligned, but given by the set of cluster centers.
- Also called “tree structured vector quantization”.
- Finding nearest neighbor to a query point involves recursively finding nearest cluster center.
- Look-up complexity O(logN)
- Also used for vocabulary quantization (see later) [Nister&Stewenius’06]

Example

3-nary tree construction:

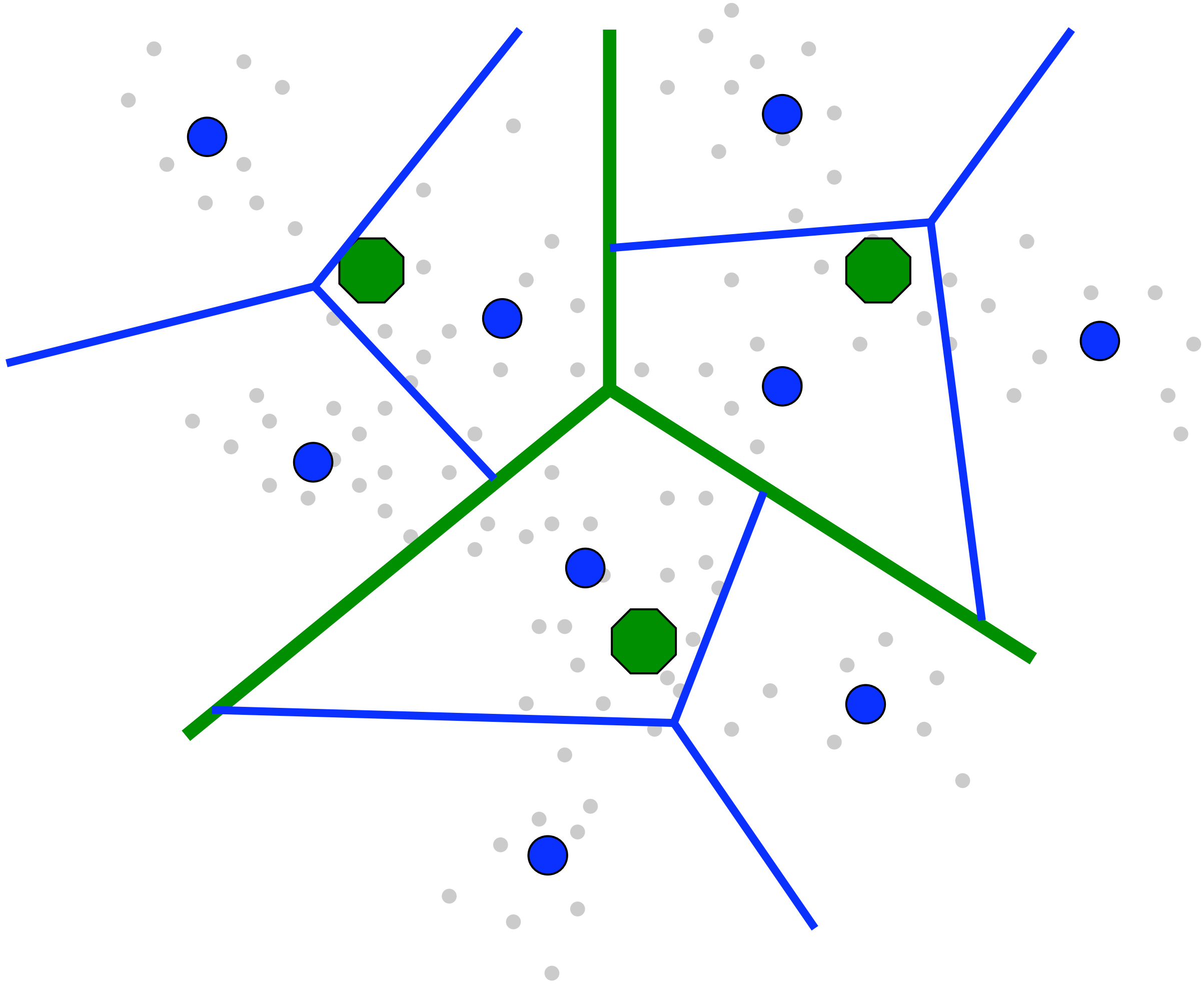


Figure credit: David Nister

Example

Query look-up:

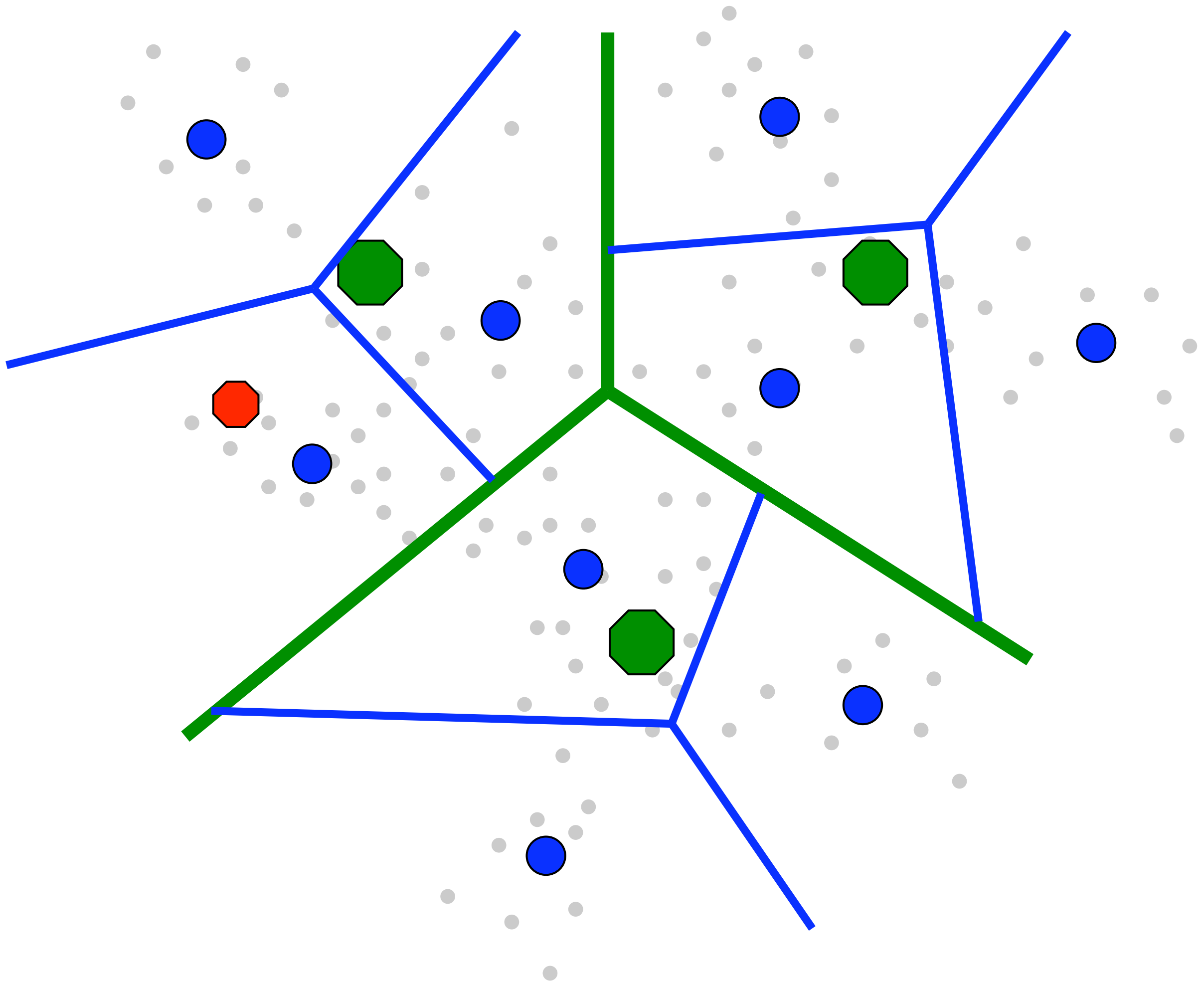
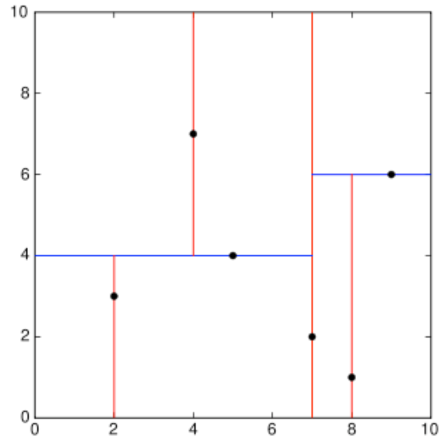
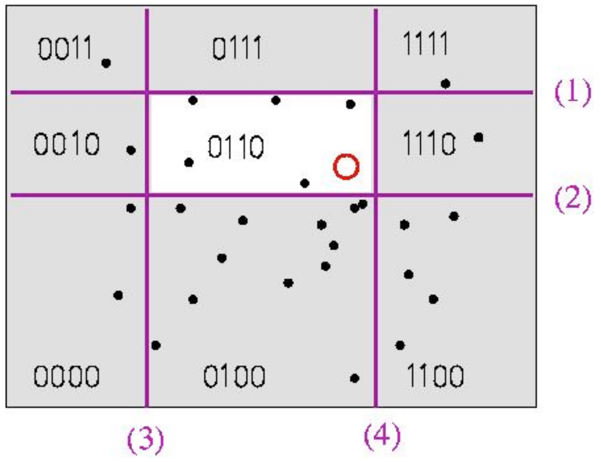


Figure credit: David Nister

Indexing local features: approximate nearest neighbor search



Best-Bin First (BBF), a variant of k-d trees that uses priority queue to examine most promising branches first [Beis & Lowe, CVPR 1997]



Locality-Sensitive Hashing (LSH), a randomized hashing technique using hash functions that map similar points to the same bin, with high probability [Indyk & Motwani, 1998]

Locality Sensitive Hashing (LSH)

Idea: construct hash functions $g: R^d \rightarrow Z^k$ such that

for any points p,q:

If $||p-q|| \leq r$, then $Pr[g(p)=g(q)]$ is “high” or “not-so-small”
If $||p-q|| > cr$, then $Pr[g(p)=g(q)]$ is “small”

Example of g: linear projections

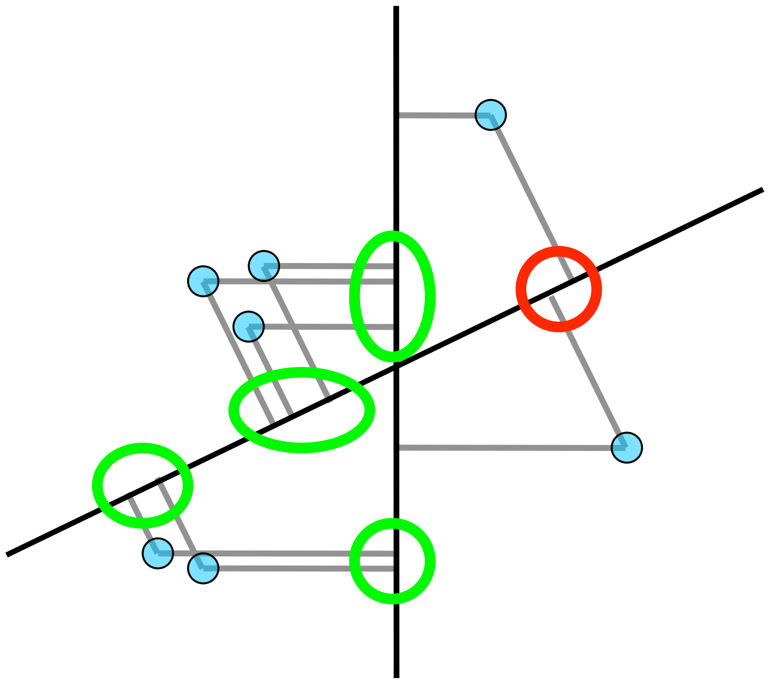
$g(p)=<h_1(p),h_2(p),\ldots,h_k(p)>$, where $h_{X,b}(p)=\lfloor (p*X+b)/w \rfloor$

$\lfloor . \rfloor$ is the “floor” operator.
X_i are sampled from a Gaussian.
w is the width of each quantization bin.
b is sampled from uniform distr. $[0,w]$.

[Datar-Immorlica-Indyk-Mirrokni’04]

Locality Sensitive Hashing (LSH)

- Choose a random projection
- Project points
- Points close in the original space remain close under the projection
- Unfortunately, converse not true



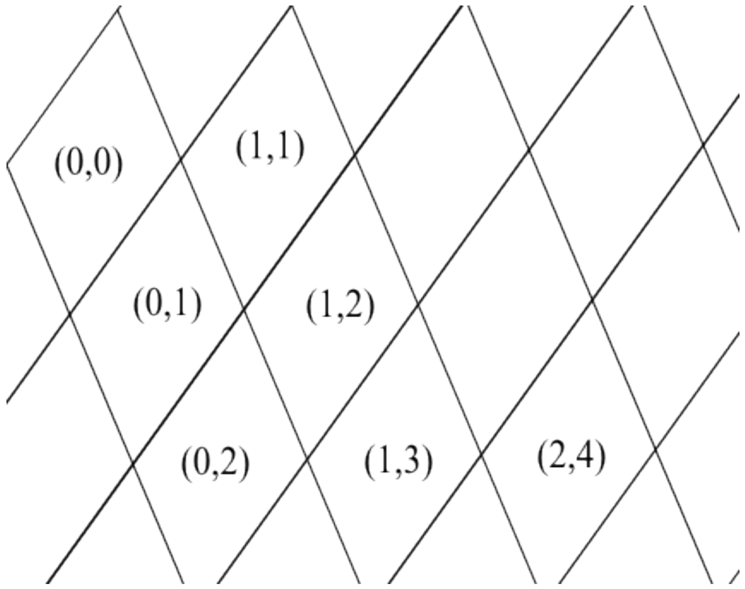
- Answer: use multiple quantized projections which define a high-dimensional “grid”

Slide: Philbin, Chum, Isard, Zissrman

Locality Sensitive Hashing (LSH)

- Cell contents can be efficiently indexed using a hash table

- Repeat to avoid quantization errors near the cell boundaries



- Point that shares at least one cell = potential candidate

- Compute distance to all candidates

Slide: Philbin, Chum, Isard, Zissrman

LSH: discussion

In theory, query time is O(kL), where k is the number of projections and L is the number of hash tables, i.e. independent of the number of points, N.

In practice, LSH has high memory requirements as large number of projections/ hash tables are needed.

Code and more materials available online:

http://www.mit.edu/~andoni/LSH/

Hashing functions could be also data-dependent (PCA) or learnt from labeled point pairs (close/far).

Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *NIPS, 2008.*

R. Salakhutdinov and G. Hinton, "Semantic Hashing," ACM SIGIR, 2007.

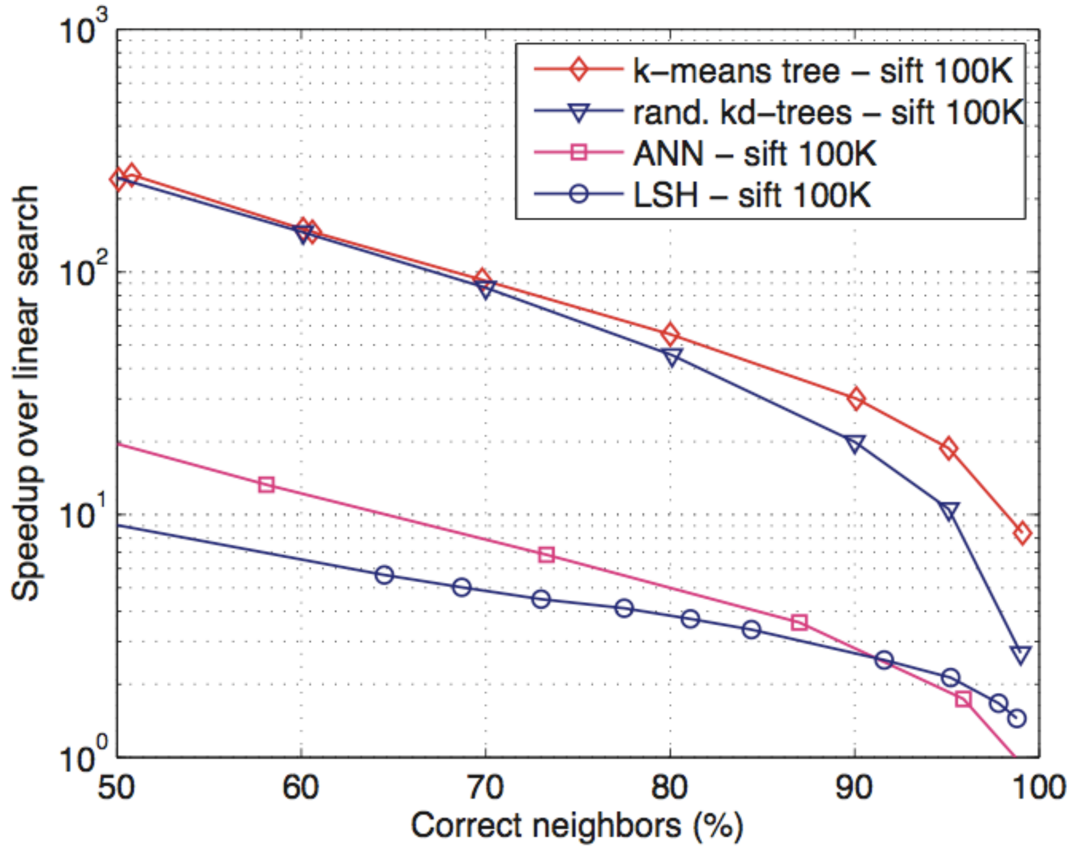
See also:

http://cobweb.ecn.purdue.edu/~malcolm/yahoo Slaney2008(LSHTutorialDraft).pdf

http://www.sanjivk.com/EECS6898/ApproxNearestNeighbors_2.pdf

Comparison of approximate NN-search methods

Dataset: 100K SIFT descriptors



Code for all methods available online, see Muja&Lowe'09

Figure: Muja&Lowe'09

Approximate nearest neighbour search (references)

J. L. Bentley. Multidimensional binary search trees used for associative searching. Comm. ACM, 18(9), 1975.

Freidman, J. H., Bentley, J. L., and Finkel, R. A. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw., 3:209–226, 1977.*

Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R., and Wu, A. Y. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *Journal of the ACM, 45:891–923, 1998.*

C. Silpa-Anan and R. Hartley. Optimised KD-trees for fast image descriptor matching. In CVPR, 2008.

M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In VISAPP, 2009.

P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proc. of 30th ACM Symposium on Theory of Computing, 1998*

G. Shakhnarovich, P. Viola, and T. Darrell, "Fast pose estimation with parameter-sensitive hashing," in *Proc. of the IEEE International Conference on Computer Vision, 2003.*

R. Salakhutdinov and G. Hinton, "Semantic Hashing," ACM SIGIR, 2007.

Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *NIPS, 2008.*

ANN - search (references continued)

O. Chum, J. Philbin, and A. Zisserman. Near duplicate image detection: min-hash and tf-idf weighting. BMVC., 2008.

M. Raginsky and S. Lazebnik, “Locality-Sensitive Binary Codes from Shift-Invariant Kernels,” in *Proc. of Advances in neural information processing systems, 2009.*

B. Kulis and K. Grauman, “Kernelized locality-sensitive hashing for scalable image search,” Proc. of the IEEE International Conference on Computer Vision, 2009.

J. Wang, S. Kumar, and S.-F. Chang, “Semi-supervised hashing for scalable image retrieval,” in IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), 2010.

J. Wang, S. Kumar, and S.-F. Chang, “Sequential projection learning for hashing with compact codes,” in Proceedings of the 27th International Conference on Machine Learning, 2010.