

Query-Adaptative Locality Sensitive Hashing

Hervé Jégou, INRIA/LJK

Laurent Amsaleg, CNRS/IRISA

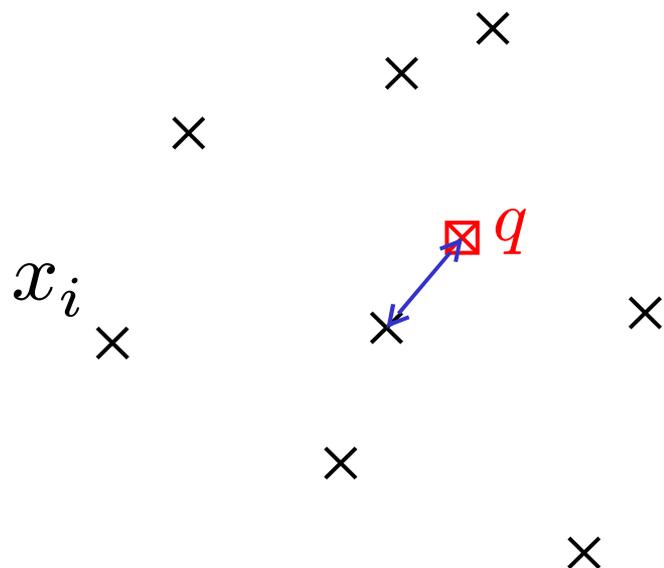
Cordelia Schmid, INRIA/LJK

Patrick Gros, INRIA/IRISA

ICASSP'2008

April 4th 2008

Problem setup



We want to find the (k-)nearest neighbor(s)
of a given query vector

→ without computing all distances!

Curse of the dimensionality

- exact search inefficient

→ approximate nearest neighbor

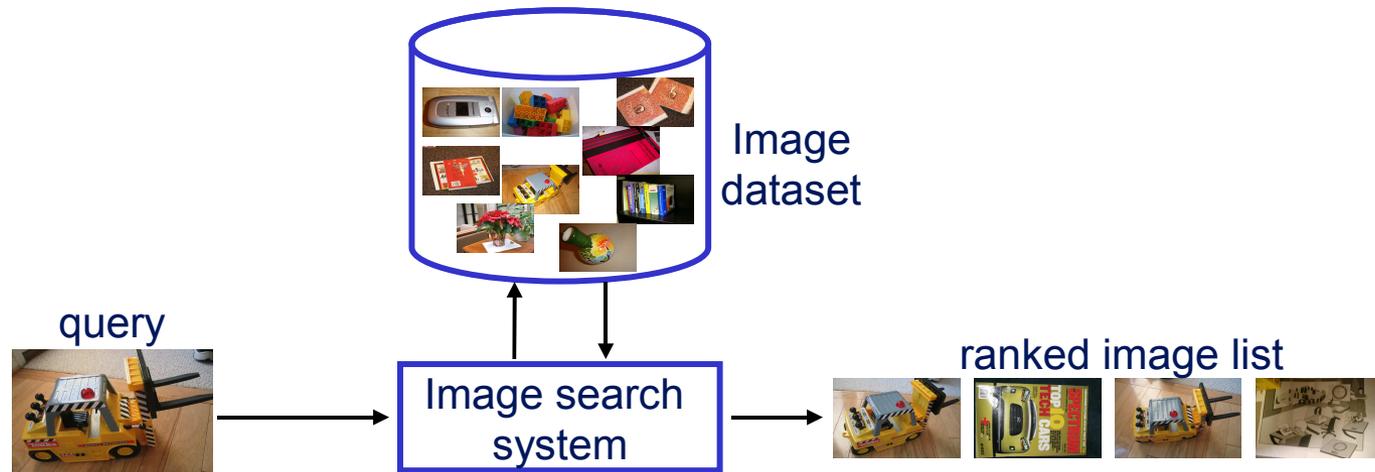
dataset: n d -dimensional vectors

$$i = 1..n, \quad x_i = (x_1, \dots, x_d)$$

query

$$q = (q_1, \dots, q_d)$$

Application : large-scale (= 1 million) image search



State-of-the-art for image search:

- local description \approx 2000 local descriptors per image
- SIFT descriptors [Lowe 04]: $d=128$, Euclidean unitary vectors

INTENSIVE USE OF NEAREST NEIGHBOR SEARCH

Approximate nearest neighbor (ANN) search

Many existing approaches

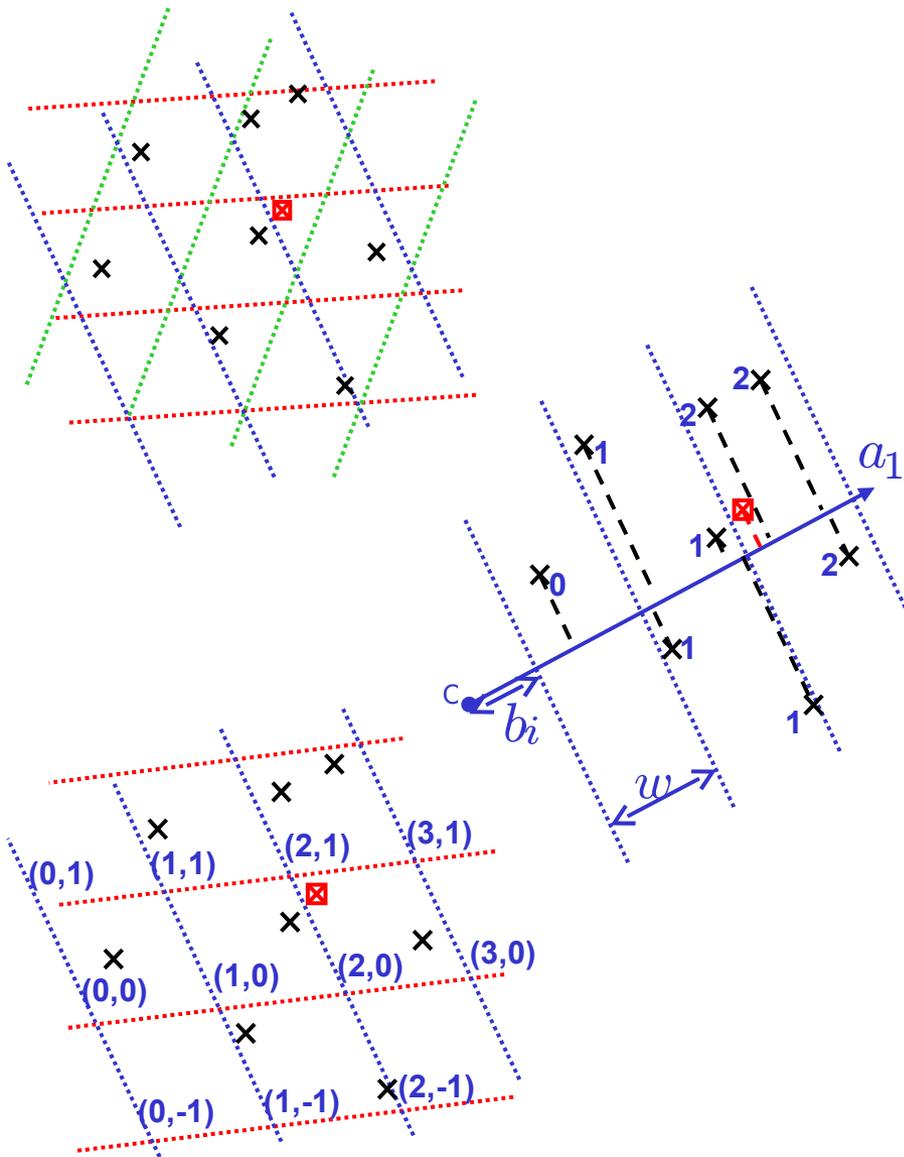
- very popular one: Locality Sensitive Hashing (LSH)
→ provides some guarantees on the search quality for some distributions

LSH: many variants, e.g.,

- for the Hamming space [Gionis, Indyk, Motwani, 99]
- Euclidean version [Datar, Indyk, Immorlica, Mirrokni, 04] → E²LSH
- using Leech lattice quantization [Andoni, Indyk, 06]
- spherical LSH [Terasawa, Tanaka, 07]

and applications: computer vision [Shakhnarovich & al, 05], music search [Casey, Slaney, 07], etc

Euclidean Locality Sensitive Hashing (E2LSH)



1) Projection on m random directions

$$h_i^r(x) = \frac{\langle x | a_i \rangle - b_i}{w}$$

$$h_i(x) = \lfloor h_i^r(x) \rfloor$$

2) Construction of ℓ hash functions:
concatenate k indexes h_i per hash
function

$$g_j(x) = (h_{j,1}(x), \dots, h_{j,k}(x))$$

3) For each g_j , compute two hash values

universal hash functions: $u_1(\cdot)$, $u_2(\cdot)$

store the vector id in a hash table

Search: algorithm summary and complexity

- For all h_i , compute $h_i(q)$ $O(m d)$
- For $j = 1..\ell$, compute $g_j(q)$ and hash values $u_1(g_j(q))$ and $u_2(g_j(q))$ $O(\ell k)$
- For $j = 1..\ell$, retrieve the vectors id having the same hash keys $O(\ell \tau n)$
 - proportion τ of the dataset vectors, i.e. $\tau * n$ vectors
- Exact distance computation between query and retrieved vectors $O(\ell \tau n d)$

Large dataset \Rightarrow step 4 is *by far* the most computationally intensive

Performance measure: **rate of correct nearest neighbors found vs average short-list size**

Geometric hash function: the lattice choice [Andoni Indyk 06]

Motivation: instead of using

$$h_i : \mathbb{R}^d \rightarrow \mathbb{Z}$$

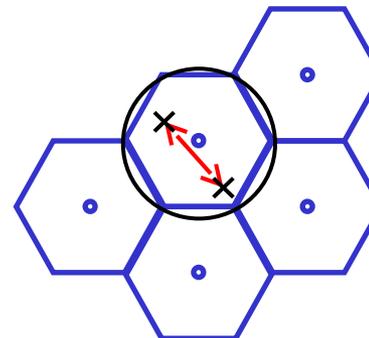
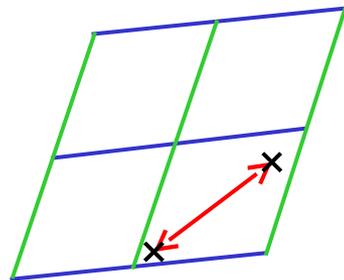
and in turn hash functions as

$$g_j(x) = (h_{j,1}(x), \dots, h_{j,k}(x))$$

Why not directly using a structured vector quantizer?

- spheres would be the best choice (but no such space partitioning)

Well-know lattice quantizers: Hexagonal (d=2), E_8 (d=8), Leech (d=24)



LSH using Lattice

Several lattices or concatenation of lattices are used for geometric hashing

$$g_j(x) = \text{lattice-idx}(x_{i,j,d^*} - b_j)$$

- b_j is now a vectorial random offset
- x_{i,j,d^*} is formed of d^* components of x (\neq for each g_j)

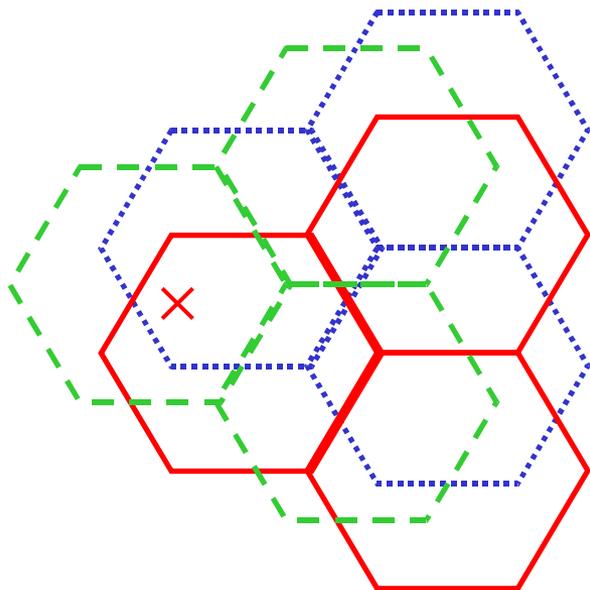
Previous work by Andoni and Indyk makes use of the Leech lattice ($d^*=24$)

- very good quantization properties
- $d^* = 24, 48, \dots$

Here, we use the E8 lattice

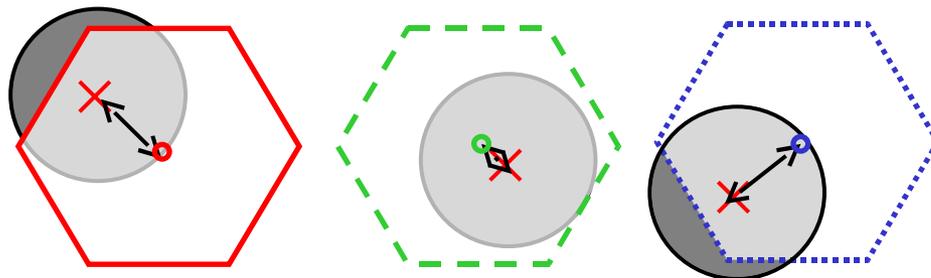
- very fast computation together with excellent quantization properties
- $d^* = 8, 16, 24, \dots$

Hash function selection criterion: motivation



Let consider several hash functions and corresponding space partitioning

The position of the query within the cell has a strong impact on the probability that vectors which are close are in the same cell or not



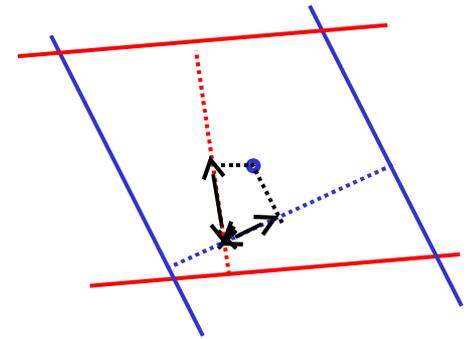
HASH FUNCTION RELEVANCE CRITERION λ_j :
the distance to the cell center in the projected k -dimensional subspace

= root square of the square Euclidean error in a quantization context

Hash function relevance criterion: E2LSH or lattice-based

E2LSH: Recall that $h_i^r(x) = \frac{\langle x | a_i \rangle - b_i}{w}$

$$h_i(x) = \lfloor h_i^r(x) \rfloor$$



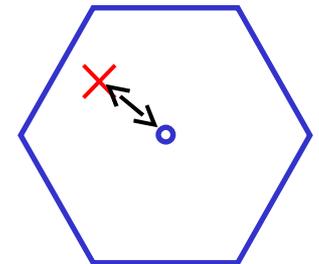
square of the relevance criterion = quantization error in the projected space

$$\lambda_j(x)^2 = \sum_{i=1..k} (h_{j,i}^r(x) - h_i(x) - 0.5)^2$$

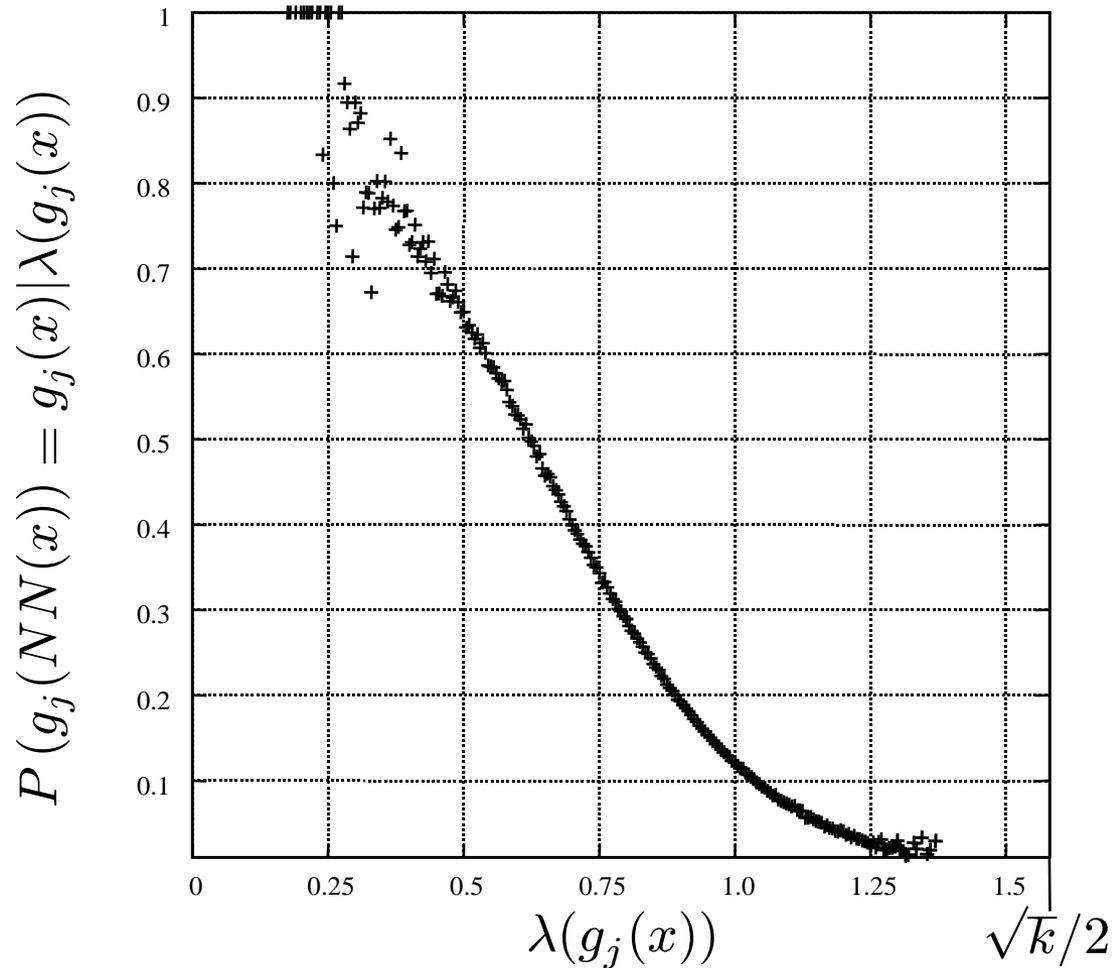
For lattice-based LSH, distance between query and lattice point

Remark for E8: λ_j requires no extra-computation

→ byproduct of the lattice point calculation



Relevance criterion: impact on quality (SIFT descriptors)



λ closer to 0: much higher confidence in the vectors retrieved

Query adaptative LSH: exploiting the criterion

Idea:

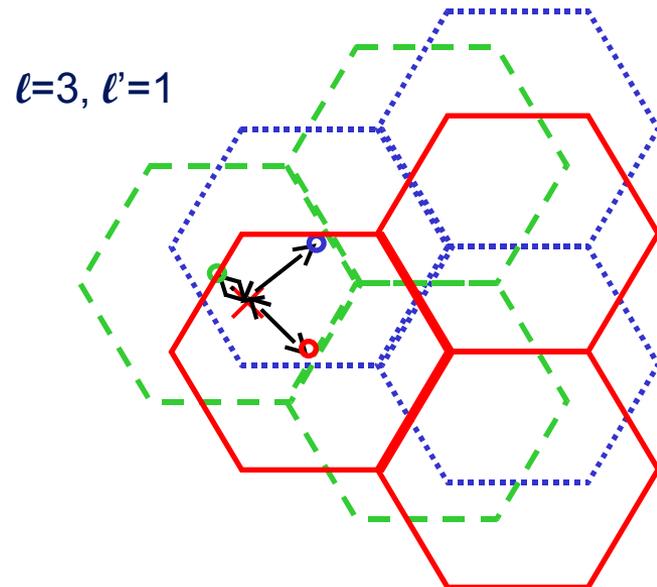
- define a larger pool of ℓ hash functions
- use only for the most relevant ones

Search is modified as follows

- for $j = 1..\ell$, compute criterion λ_j
- select the ℓ' ($\ll \ell$) hash functions associated with the lowest values of λ_j

Perform the final steps as in standard LSH, using the hash function subset only

- compute u_1 and u_2 and parse the corresponding buckets
- compute the exact distances between query and vectors retrieved from buckets



Query adaptative LSH: exploiting the criterion

Idea:

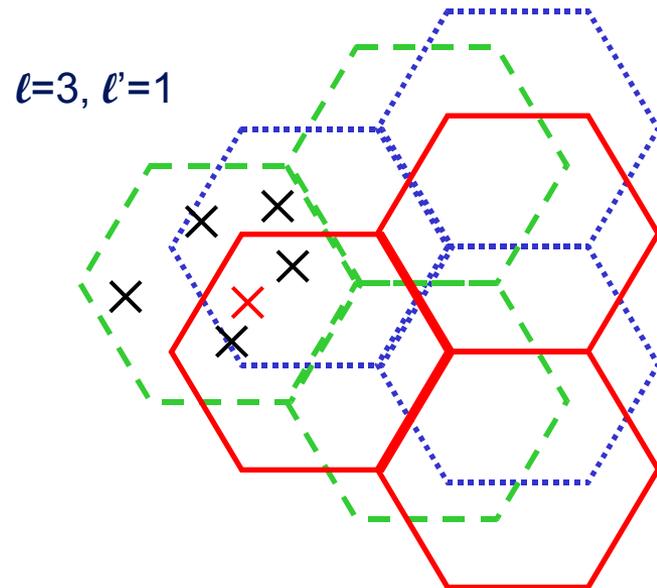
- define a larger pool of ℓ hash functions
- use only for the most relevant ones

Search is modified as follows

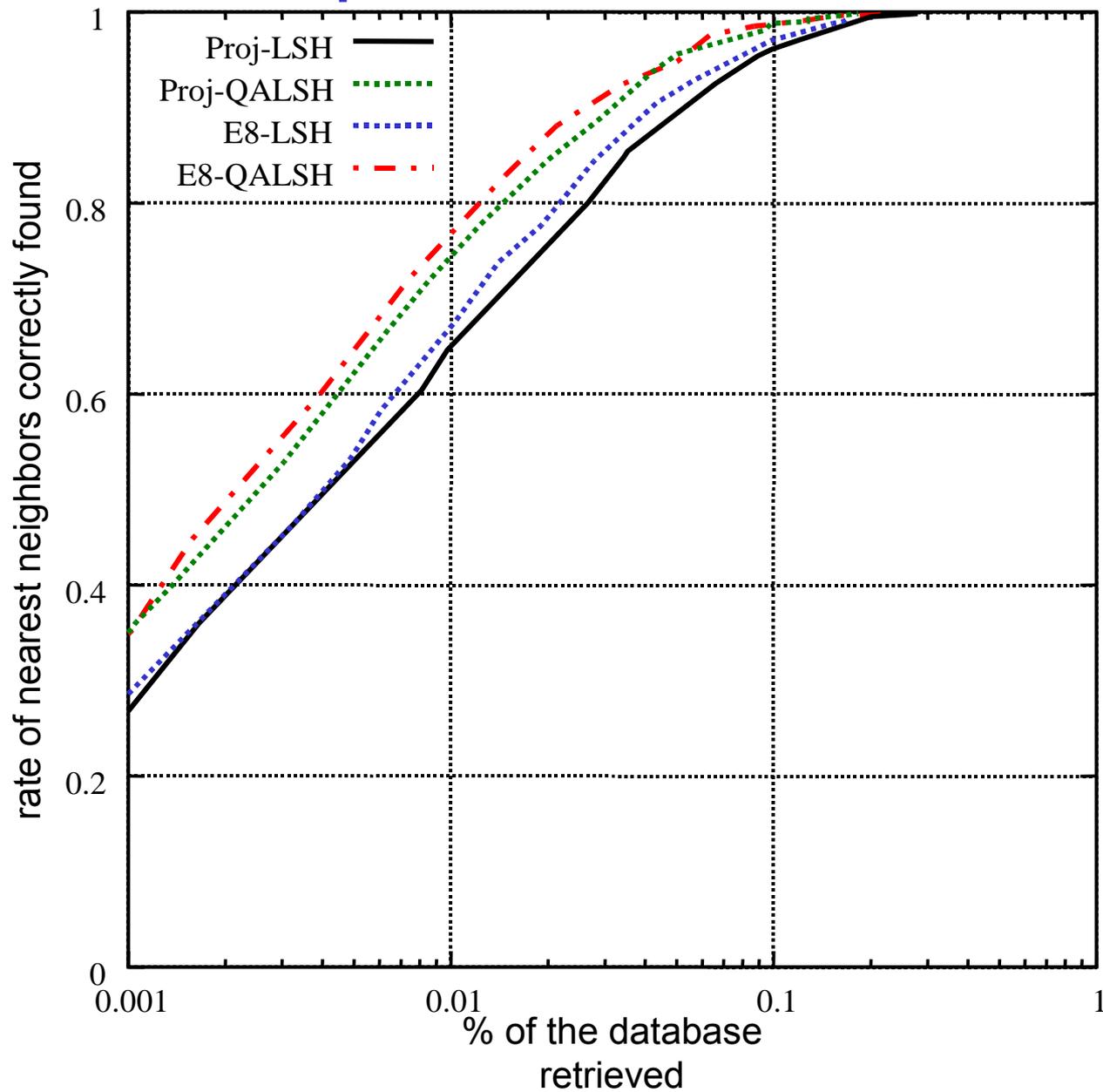
- for $j = 1..\ell$, compute criterion λ_j
- select the ℓ' ($\ll \ell$) hash functions associated with the lowest values of λ_j

Perform the final steps as in standard LSH, using the hash function subset only

- compute u_1 and u_2 and parse the corresponding buckets
- compute the exact distances between query and vectors retrieved from buckets



Results: SIFT descriptors



Conclusion

Using E8 Lattice for LSH provides

- excellent quantization properties
- high flexibility for d^*

QALSH trades memory against accuracy

→ without noticeably increasing search complexity for large datasets

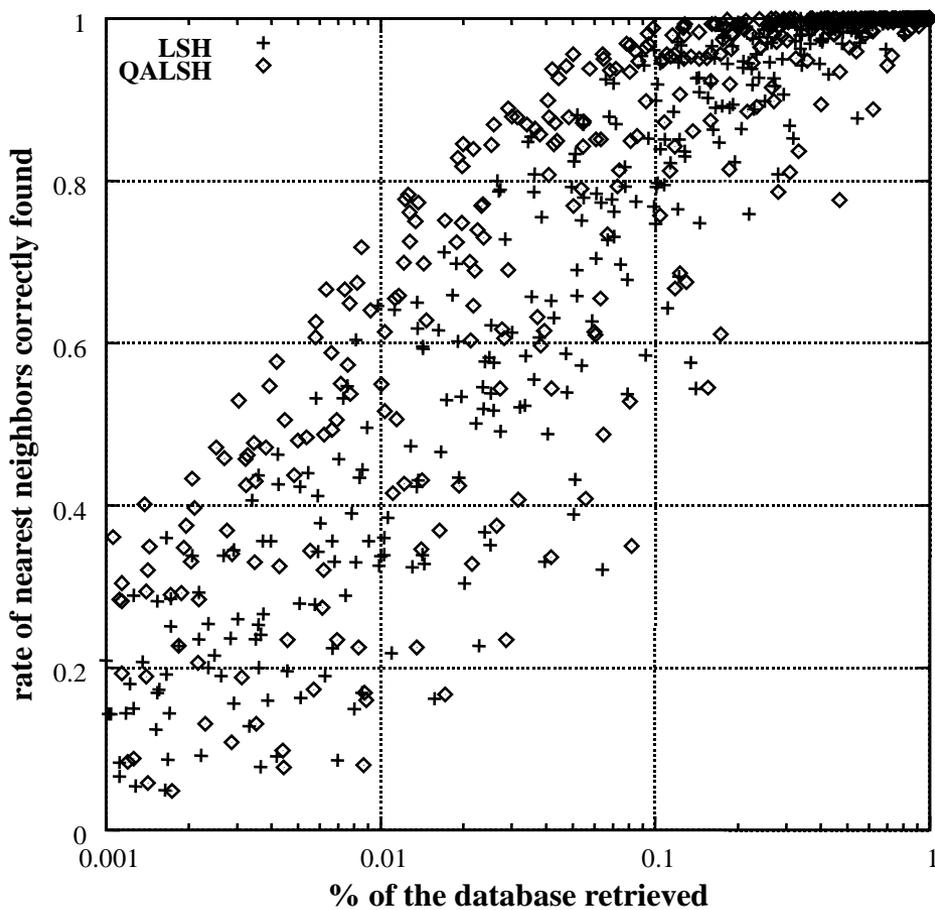
This a quite generic approach: can be jointly used with other versions of LSH

- binary or spherical LSH

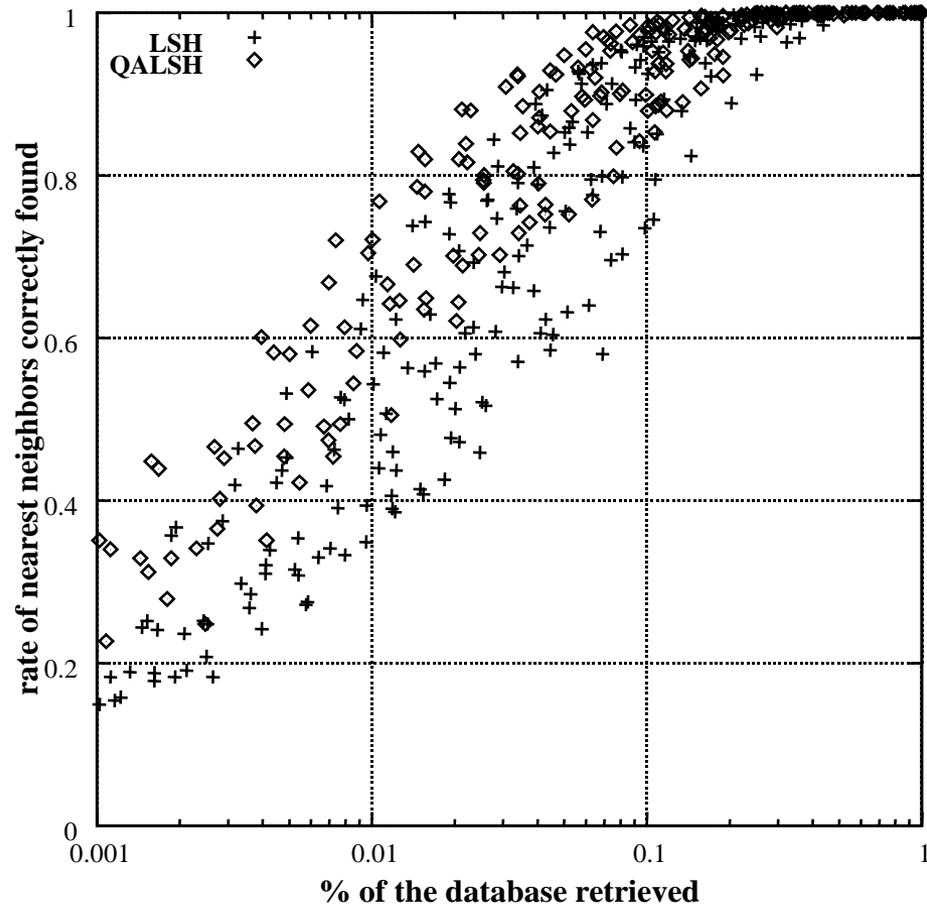
Thank you for your attention!

?

Brute force search of optimal parameters

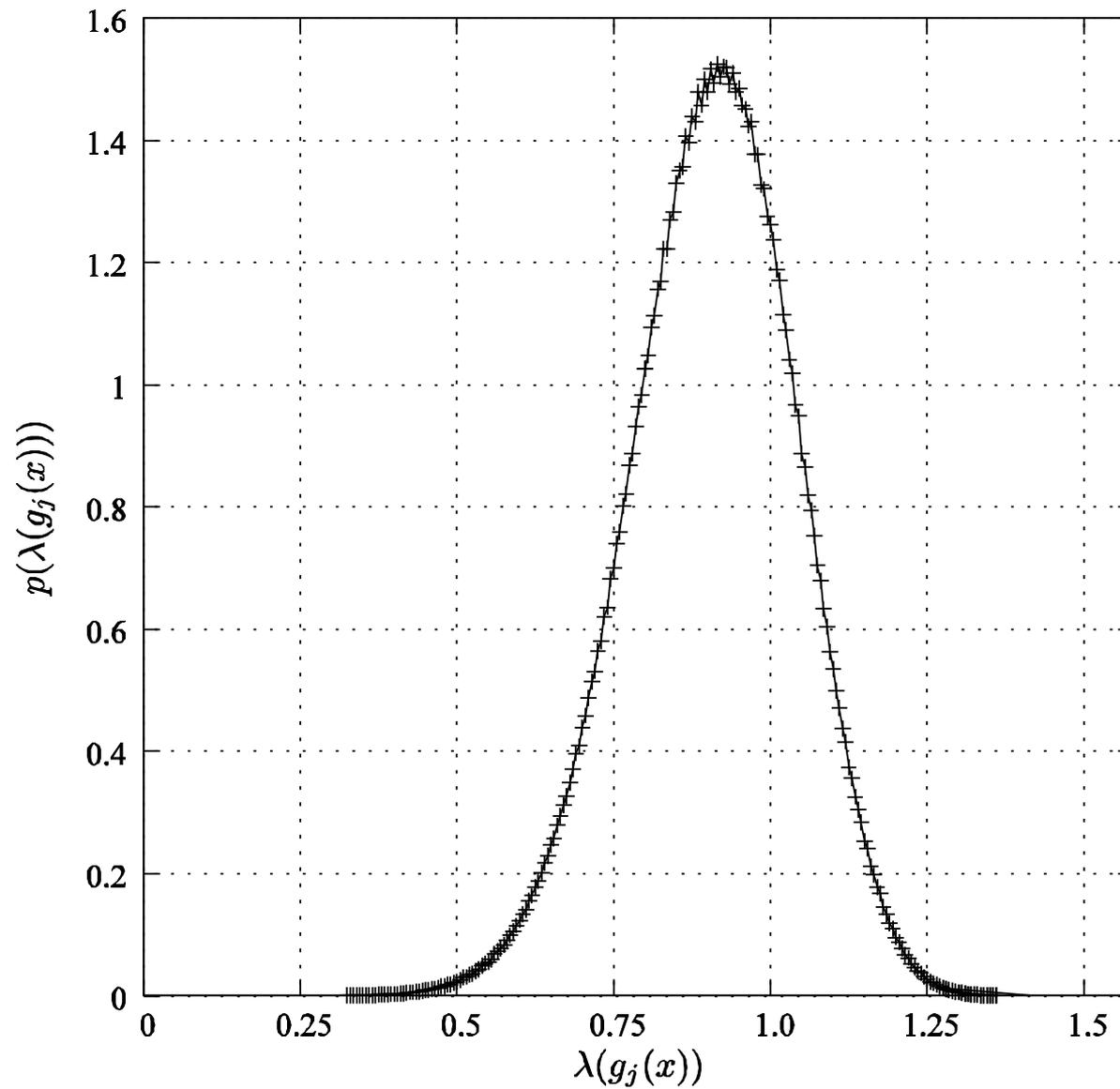


Random projection LSH

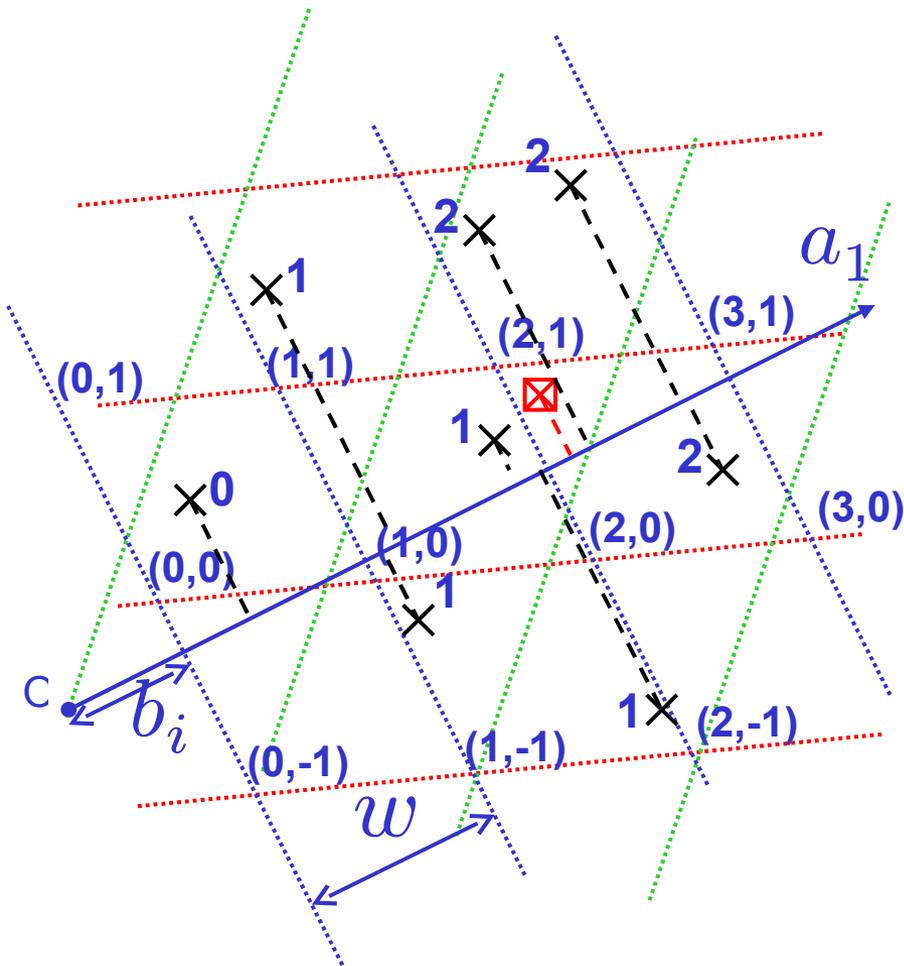


E₈ LSH

p.d.f. of the relevance criterion



Euclidean Locality Sensitive Hashing (E2LSH)



1) Projection on m random directions

$$h_i^r(x) = \frac{\langle x | a_i \rangle - b_i}{w}$$

$$h_i(x) = \lfloor h_i^r(x) \rfloor$$

2) Construction of ℓ hash functions:
concatenate k indexes h_i per hash
function

$$g_j(x) = (h_{j,1}(x), \dots, h_{j,k}(x))$$

3) For each g_j , compute two hash values

universal hash functions: $u_1(\cdot)$, $u_2(\cdot)$

store the vector id in a hash table