UFMG

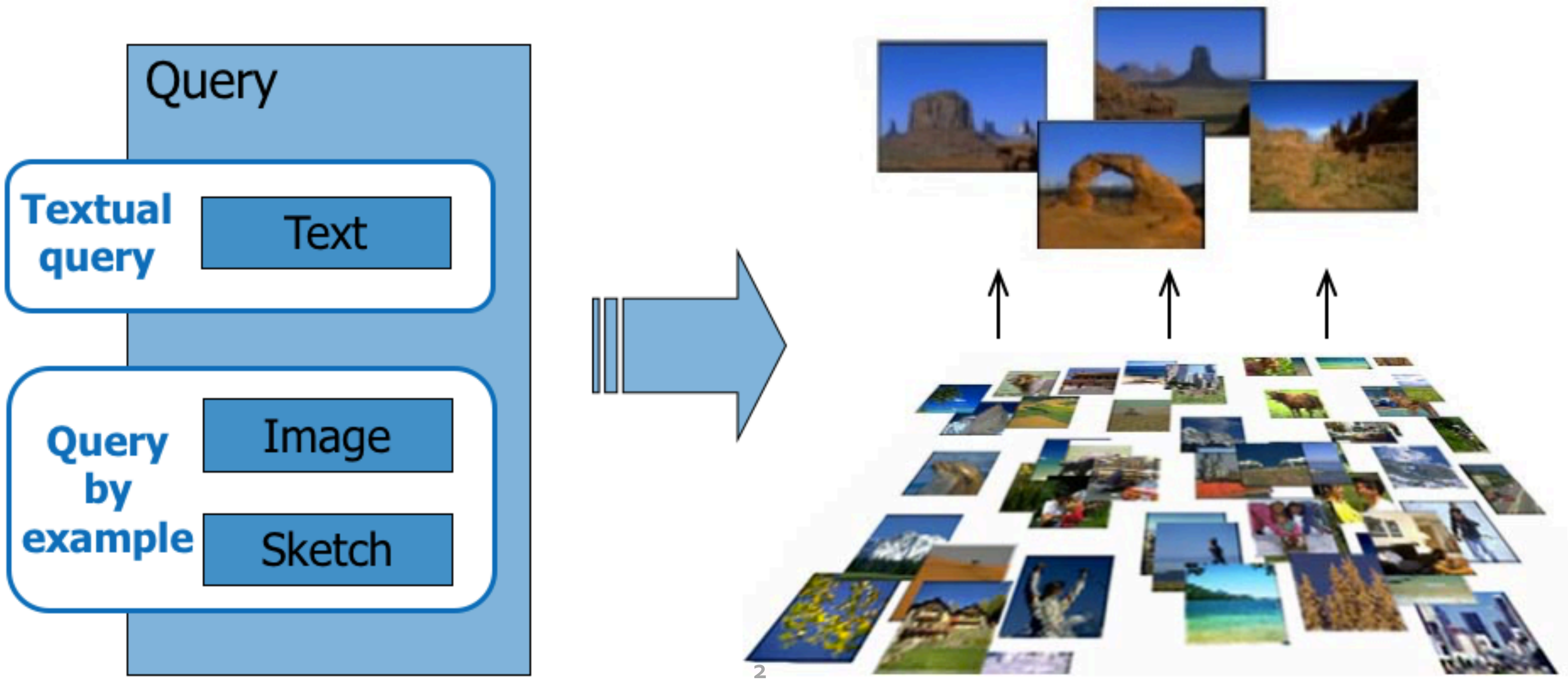
PATREO — Pattern Recognition for Earth Observation — DCC - UFMG

Locality Sensitive Hashing (LSH)

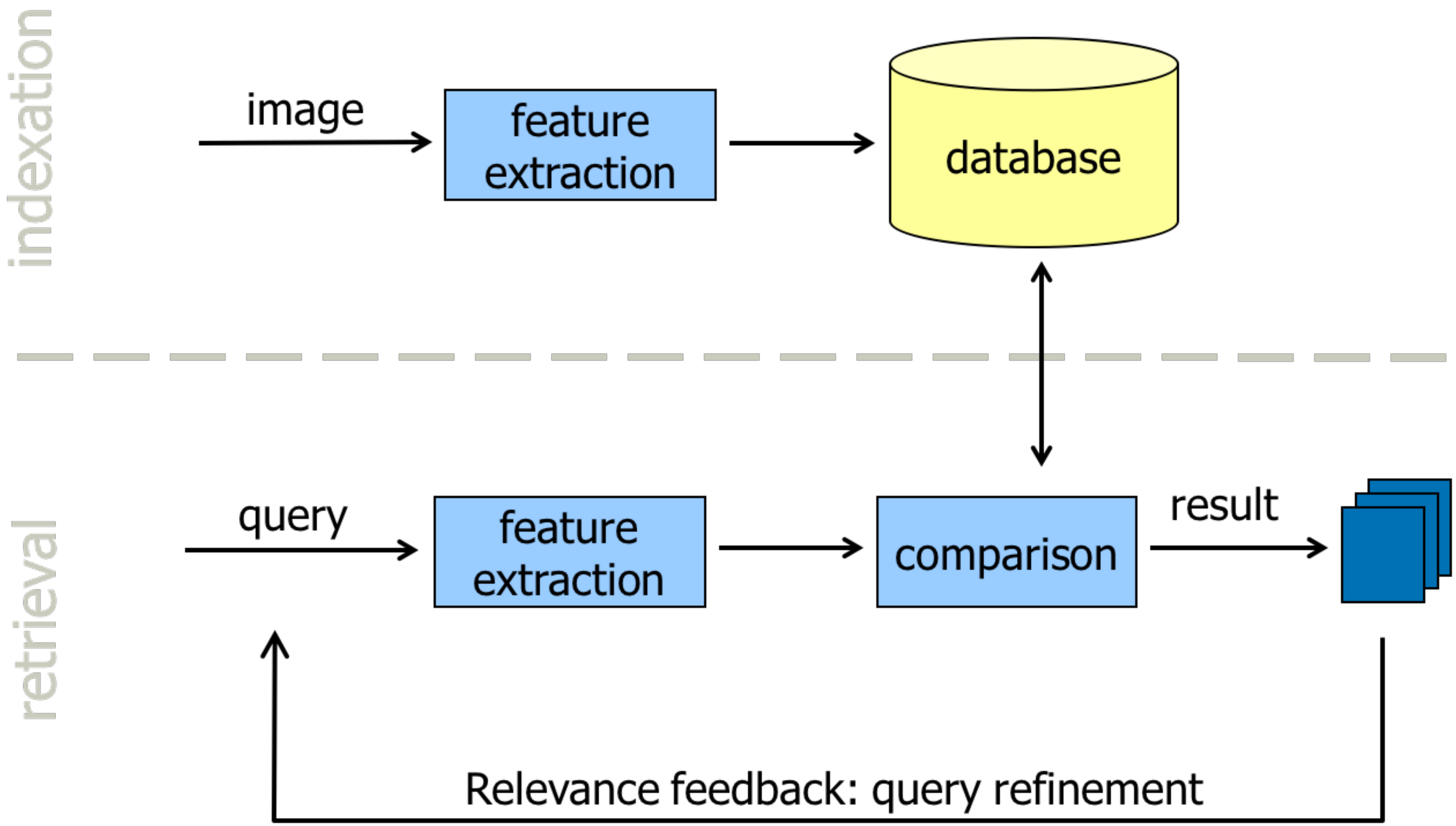
Eduardo Tavares

Motivation

- ✓ Description Based Image Retrieval (DBIR)
- ✓ Content Based Image Retrieval (CBIR)

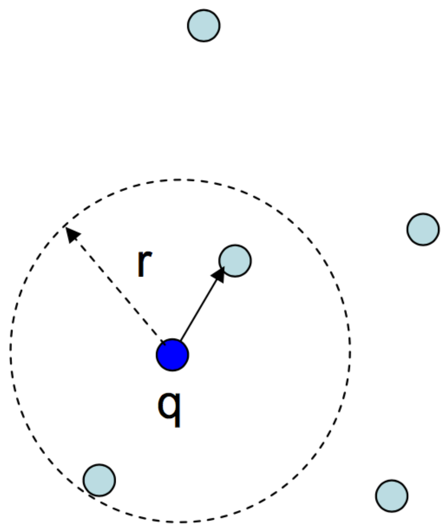


Common components of CBIR system



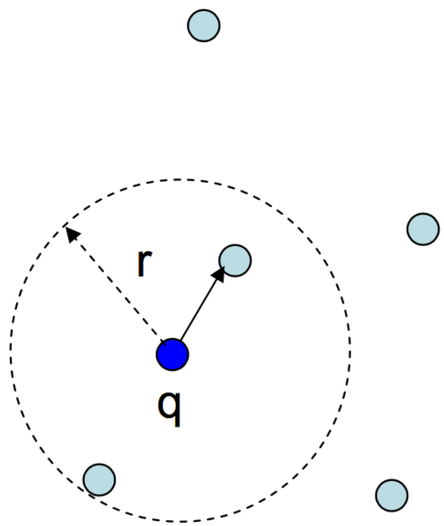
Nearest Neighbour Problem

- ✓ Given: a set P of point in R^d
- ✓ **Nearest Neighbour**: for any query q, returns a point $p \in P$ minimizing D(p,q)
- ✓ **r-Near Neighbour**: for any query q, returns a point $p \in P$ such that $D(p,q) \leq r$ (if it exists)



Nearest Neighbour Problem

- Given: a set P of point in R^d
- **Nearest Neighbour**: for any query q, returns a point p ∈ P minimizing D(p,q)
- **r-Near Neighbour**: for any query q, returns a point p ∈ P such that D(p,q) ≤ r (if it exists)

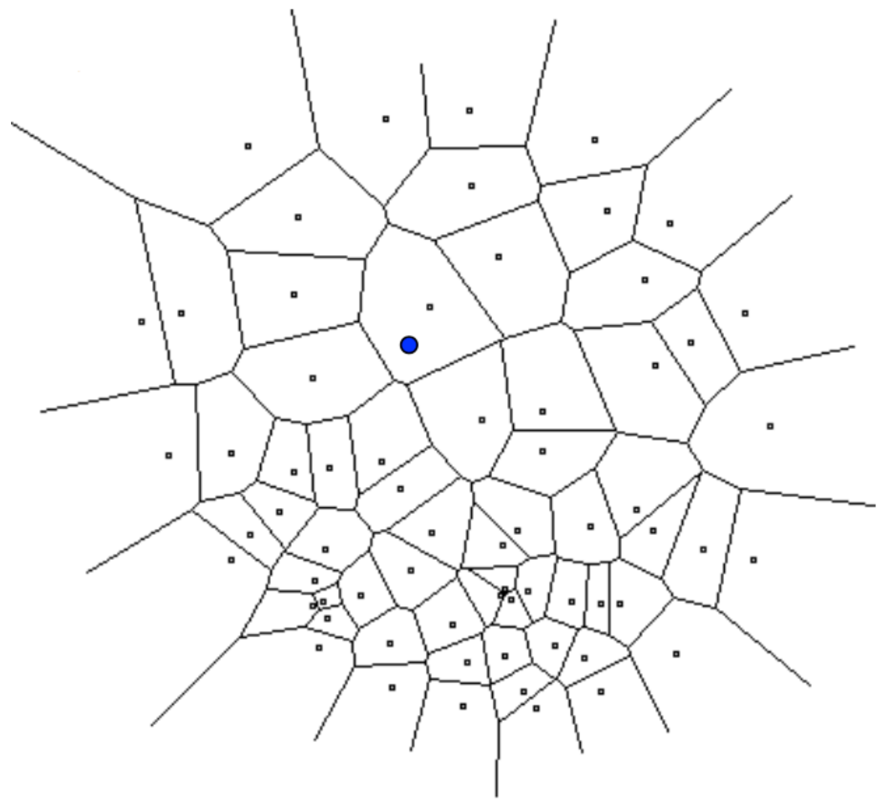


DEFINITION (ℓ_p DISTANCE). *The distance between any two d-dimensional points $\vec{o}$ and $\vec{q}$ in the ℓ_p space, denoted as $\ell_p(\vec{o}, \vec{q})$, is computed as:*

$$\ell_p(\vec{o}, \vec{q}) = \sqrt[p]{\sum_{i=1}^{d} |o_i - q_i|^p}$$

The case of d = 2

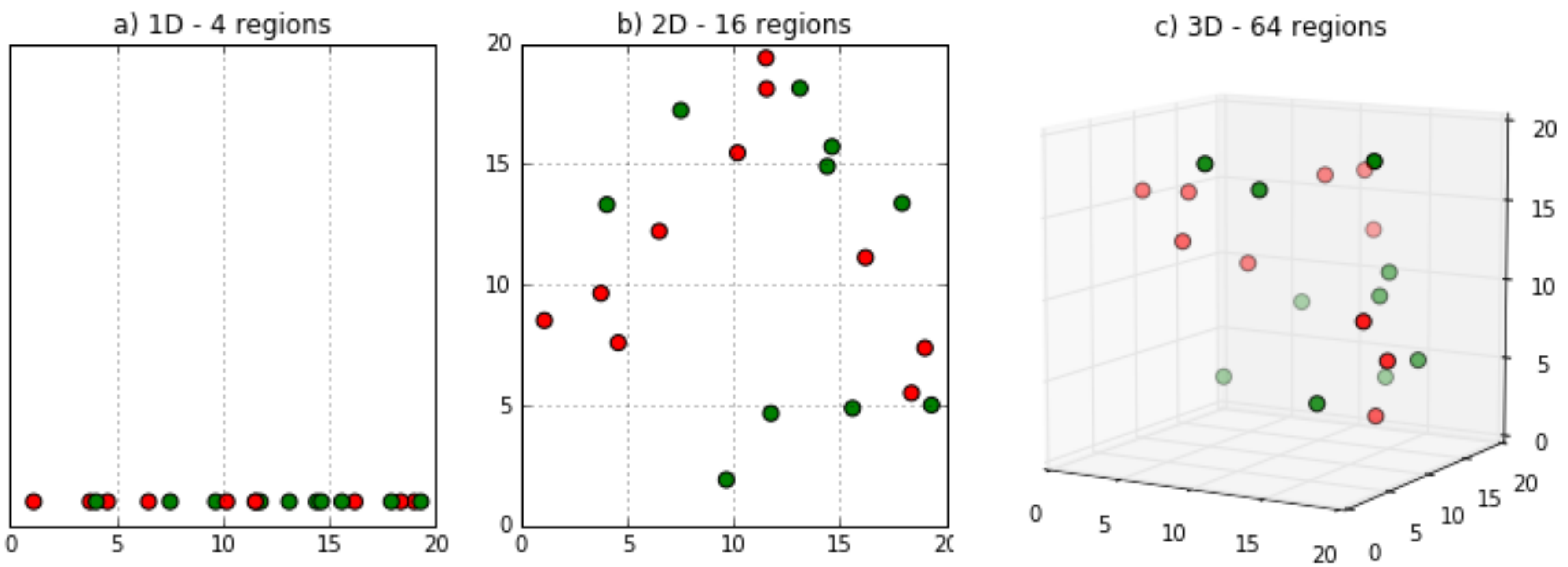
- ✓ Compute **Voronoi diagram**
- ✓ Given q, perform point location
- ✓ Performance:
  - ✓ Space: O(n)
  - ✓ Query time: O(log n)



The case of d > 2

- Voronoi diagram has size $O(n^d)$
- Another possibility: linear scan ($O(dn)$ time)
- That's all that is known about exact algorithms with theoretical guarantees.
- In practice:
  - Kd-trees work "well" in "low-medium" dimensions: require sublinear time and near linear space for d < 10 - 20
  - Time or space requirements grow exponentially in the dimension.

Problems: curse of dimensionality



The cost of exact matching

- Finding the 10-NN of 1000 distinct queries in 1 million vectors
  - Assuming 128-D Euclidian descriptors
  - i.e., 1 billion distances, computed on a 8-core machine

The cost of exact matching

- Finding the 10-NN of 1000 distinct queries in 1 million vectors
  - Assuming 128-D Euclidian descriptors
  - i.e., 1 billion distances, computed on a 8-core machine
- 5.5 seconds

- Hamming distance: 1000 queries, 1M database vectors:
  - Computing the 1 billion distances: 0.6 second

Hamming Space

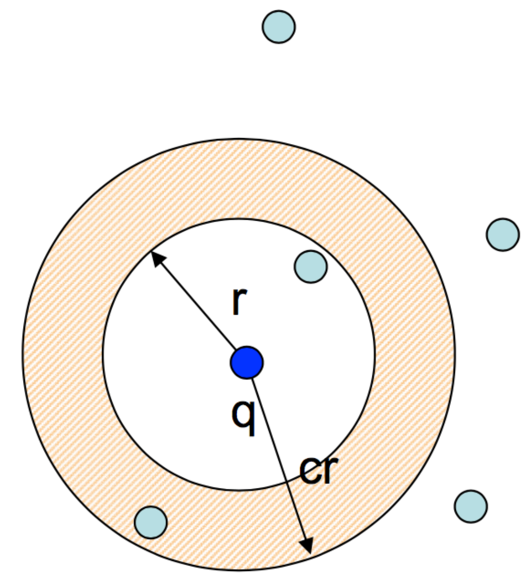
- **Definition:** *Hamming space* is the set of all 2^N binary strings of length N.
- **Definition:** The *Hamming distance* between two equal length binary strings is the number of positions for which the bits are different.
- $||1011101; 1001001||_H = 2$
- $||1110101; 1111101||_H = 1$

The cost of exact matching

- To improve the scalability: find the nearest neighbour in probability only: **Approximate nearest neighbour (ANN) search**
- An approximate nearest neighbour should suffice in most cases.

Approximate Near Neighbour

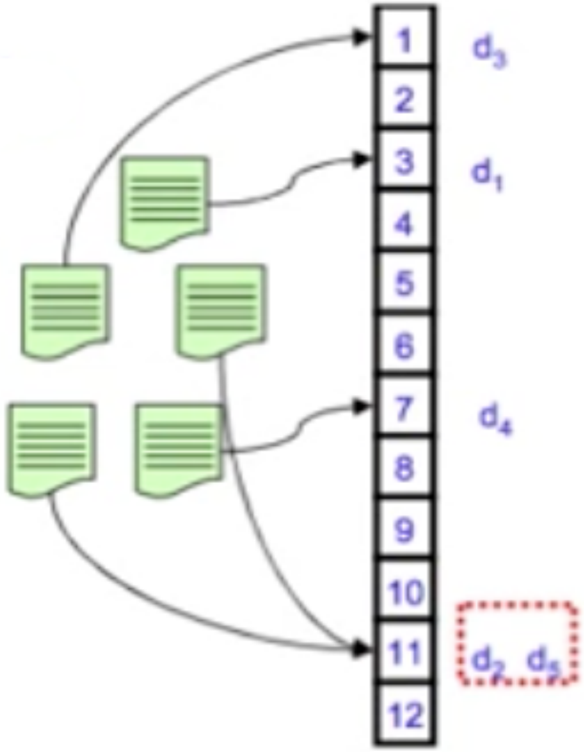
- c-Approximate r-Near Neighbour: build data structure which, for any query q:
  - If there is a point $p \in P$, $||p - q|| \leqq r$
  - it returns $p' \in P$, $||p' - q|| \leqq cr$



- Three (contradictory) performance criteria for ANN schemes:
  - Search quality (retrieved vectors are actual nearest neighbours)
  - Speed
  - Memory usage

Hashing

- Hash function: any function $\mathcal{H}$ which has, as a minimum, the following two properties:
  - Compression
  - ease of computation



Locality Sensitive Hashing

- Idea: project the data into a low-dimensional binary (Hamming) space; while preserving some properties from the original space.

- Construct hash functions $h: R^d \rightarrow U$ such that for any points p, q:
  - If $D(p,q) \leq r$, then $Pr[h(p) = h(q)]$ is "high".
  - If $D(p,q) > cr$, then $Pr[h(p) = h(q)]$ is "small".

- Then we can solve the problem by performing hash table lookup.
- LSH is a general framework; for a given D we need to find the right h.

LSH [Indyk-Motwani'98]

- A family $\mathcal{H}$ of functions $h: R^d \rightarrow U$ is called (P_1, P_2, r, cr)-sensitive , if for any p, q:
  - If $D(p,q) < r$, then $Pr[h(p) = h(q)] > P_1$
  - If $D(p,q) > cr$, then $Pr[h(p) = h(q)] < P_2$.
- $P_1 > P_2$.

Bit sampling

- ✓ Works for the Hamming distance over d-dimensional vectors $\{0,1\}^d$.
- ✓ The family $\mathcal{H}$ of hash functions is simply the family of all the projections of points in one of the d coordinates.
- ✓ $\mathcal{H} = \{h : \{0,1\}^d \rightarrow \{0,1\} \mid h(x) = x_i$ for some $i \in \{1, \ldots, d\}$, where x_i is the *i*th coordinate of x.
- ✓ A random function h from $\mathcal{H}$ simply selects a random bit from the input point.

Algorithm: preprocessing

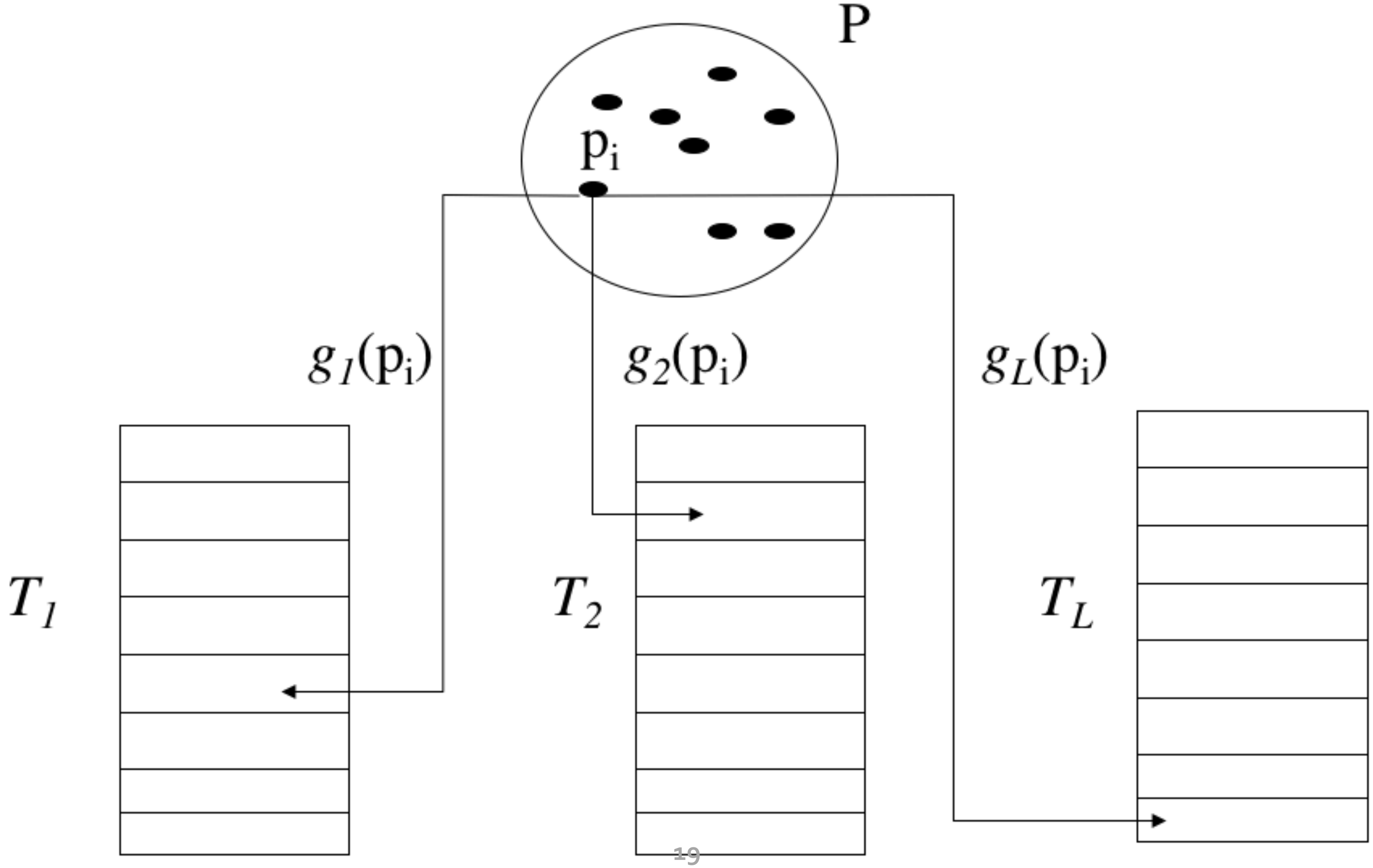
- Hash the data-point using several LSH functions ($g_i(.) = \{ h_1(.) \ldots h_k(.) \}$) so that probability of collision is higher for closer objects

- Input
  - Set of N points $\{ p_1, \ldots\ldots\ldots p_n \}$
  - L (number of hash tables)
- Output
  - Hash tables T_i , $i = 1, 2, \ldots. L$
- Foreach $i = 1, 2, \ldots. L$
  - Initialize T_i with a random hash function $g_i(.)$
- Foreach $i = 1, 2, \ldots. L$

  Foreach $j = 1, 2, \ldots. N$

  Store point p_j on bucket $g_i(p_j)$ of hash table T_i

Algorithm: preprocessing



Algorithm : cr - NNS Query

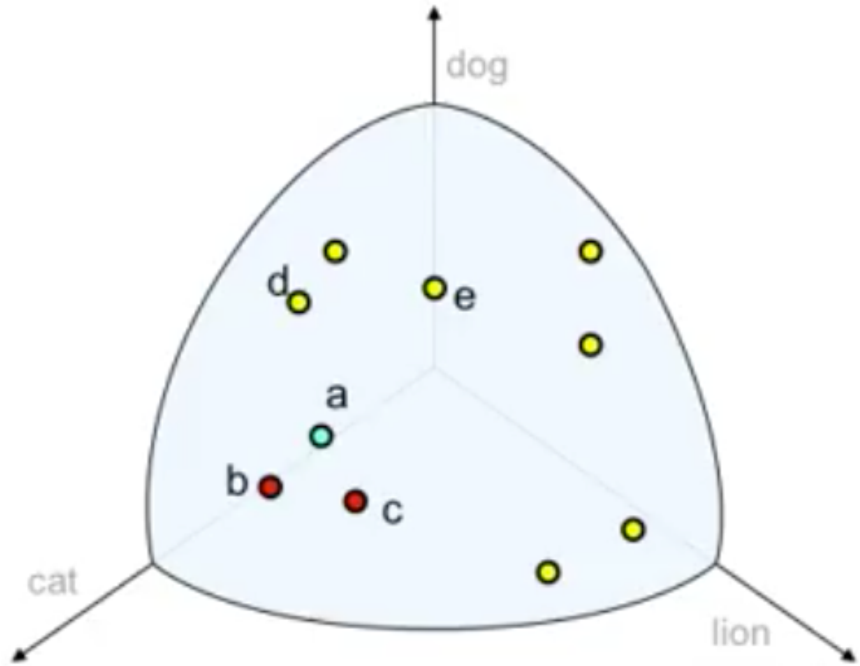
- Input
  - Query point q
  - K (number of approx. nearest neighbors)
- Access
  - Hash tables T_i , $i = 1$, 2, …. L
- Output
  - Set S of K (or less) approx. nearest neighbors
- S $\leftarrow \varnothing$

  Foreach $i = 1$, 2, …. L
  - S $\leftarrow$ S $\cup$ { points found in $g_i(q)$ bucket of hash table T_i }

Random hyperplane based LSH

- ✓ Selects b hash functions $h_r(\cdot)$, each of which simply rounds the output of the product of x with a random hyperplane defined by a random vector r:

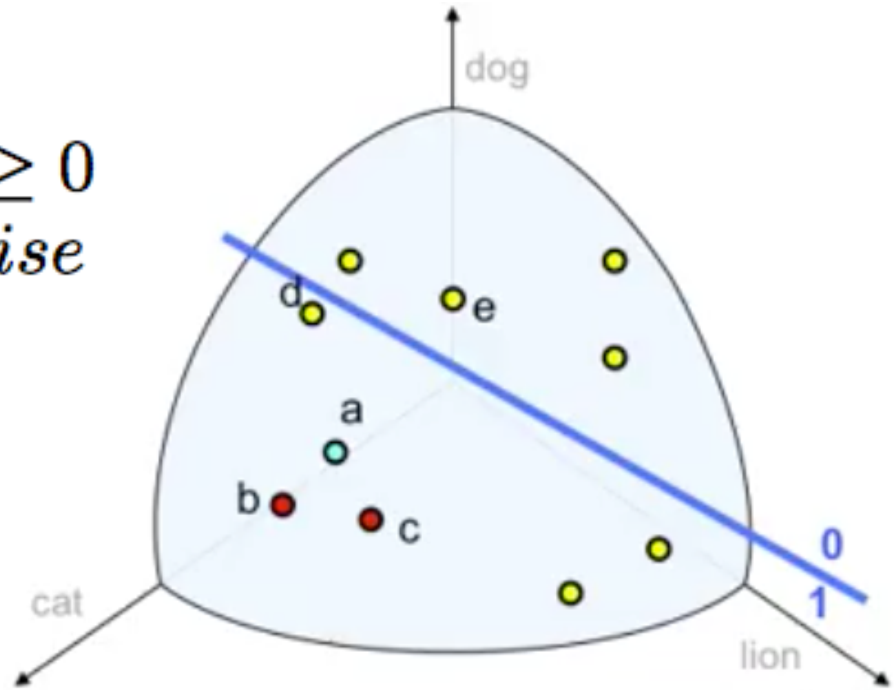
$$h_r(x) = \begin{cases} 1 & if\ r^T x \geq 0 \\ 0 & otherwise \end{cases}$$



Random hyperplane based LSH

- Selects b hash functions $h_r(\cdot)$, each of which simply rounds the output of the product of x with a random hyperplane defined by a random vector r:

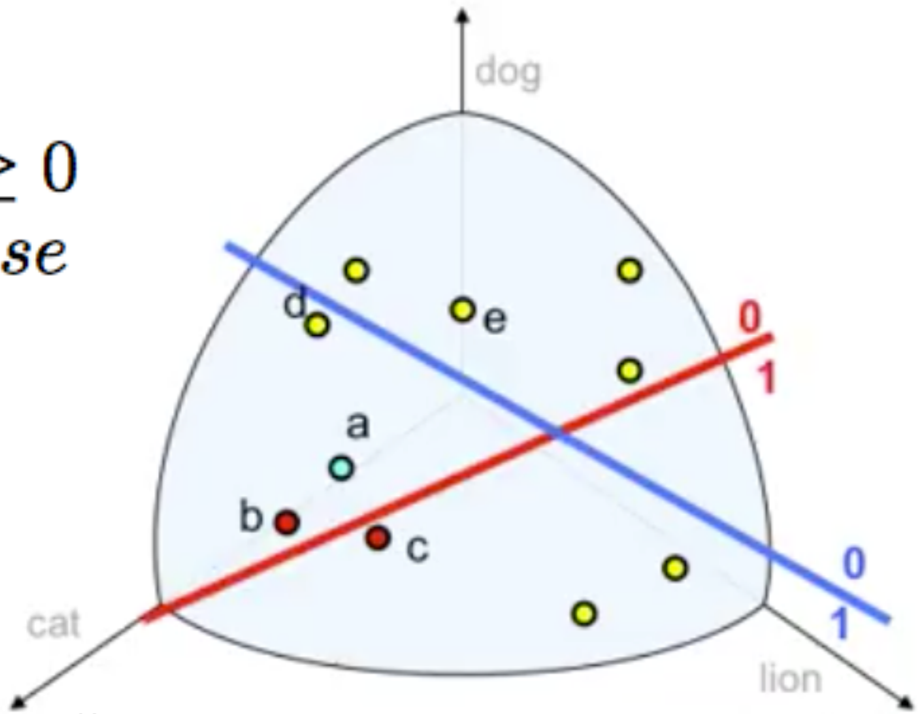
$$h_r(x) = \begin{cases} 1 & if\ r^T x \geq 0 \\ 0 & otherwise \end{cases}$$



Random hyperplane based LSH

- Selects b hash functions $h_r(\cdot)$, each of which simply rounds the output of the product of x with a random hyperplane defined by a random vector r:

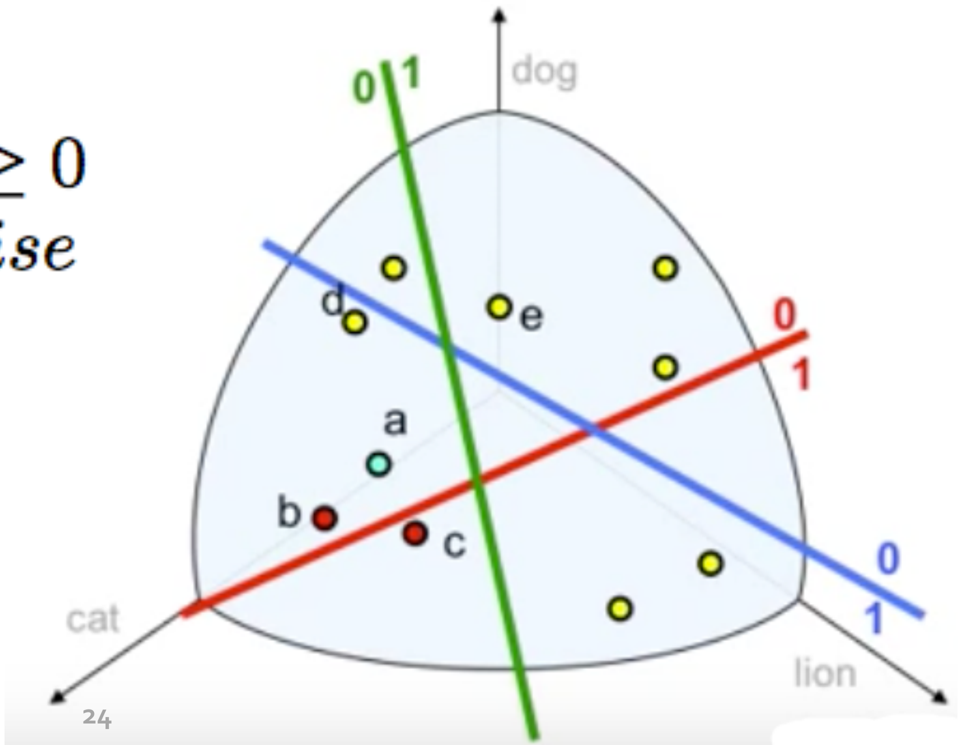
$$h_r(x) = \begin{cases} 1 & if\ r^T x \geq 0 \\ 0 & otherwise \end{cases}$$



Random hyperplane based LSH

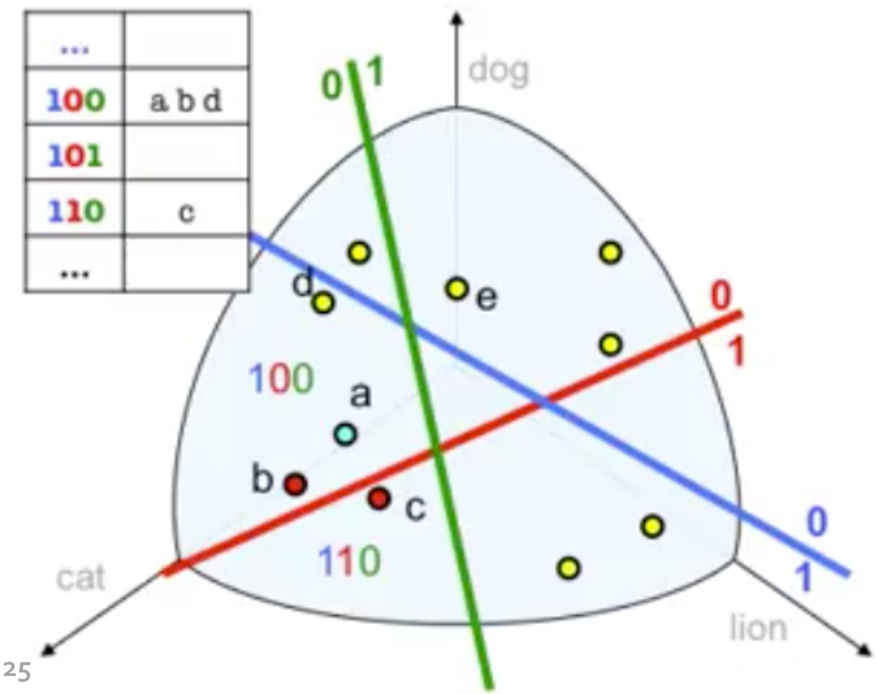
- ✓ Selects b hash functions $h_r(\cdot)$, each of which simply rounds the output of the product of x with a random hyperplane defined by a random vector r:

$$h_r(x) = \begin{cases} 1 & if\ r^T x \geq 0 \\ 0 & otherwise \end{cases}$$



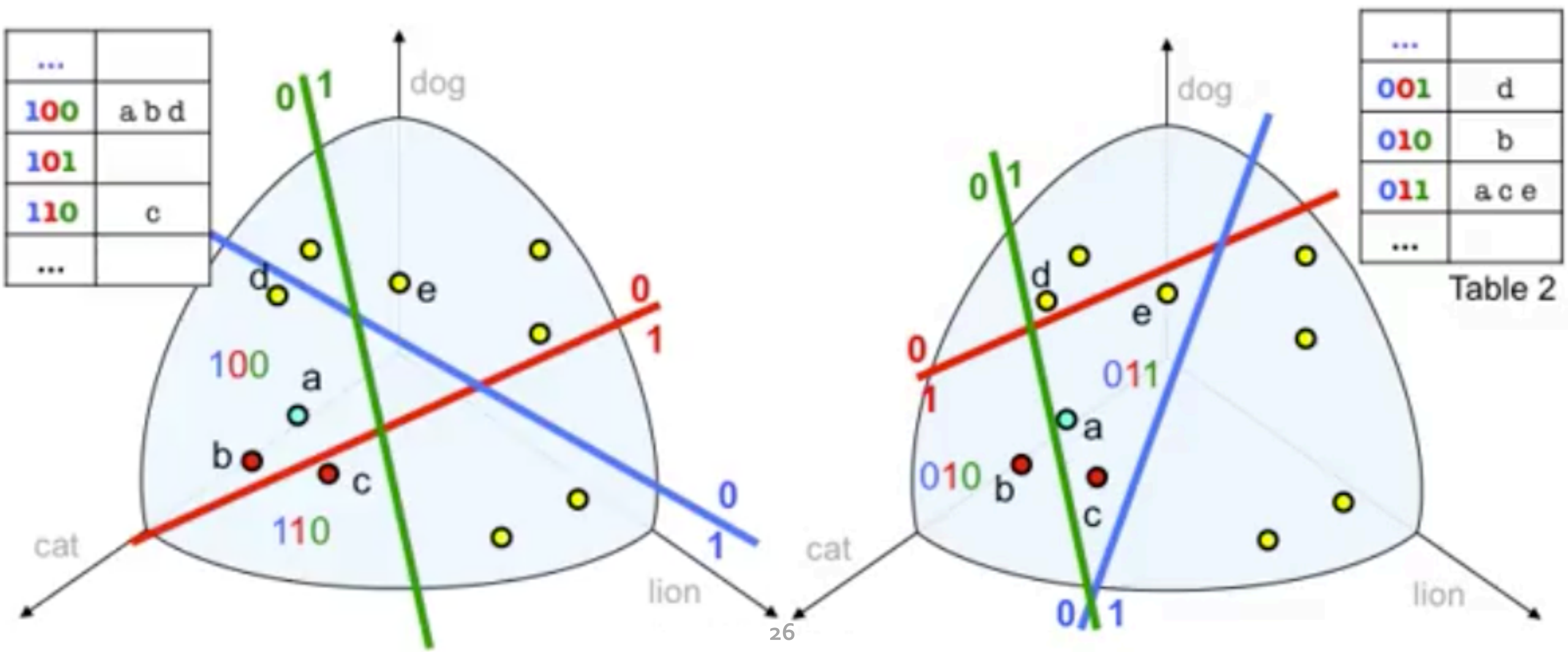
Random hyperplane based LSH

- Compare **a** to points with same hash-code
  - **b** … indeed similar to a
  - **d** … false positive, will be eliminated
  - **c** … different hash-code, will miss it



Random hyperplane based LSH

- ✓ Repeat with different hyperplanes

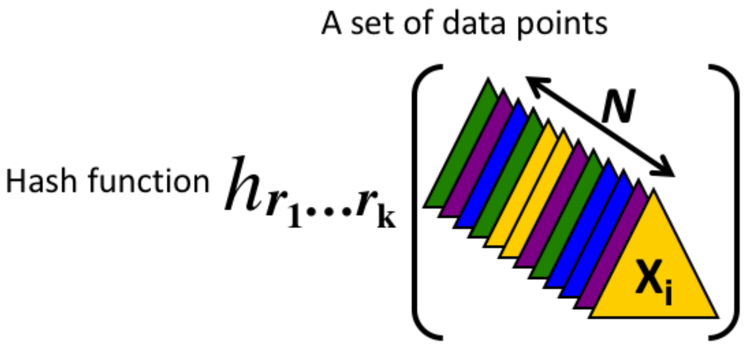


...	
100	a b d
101	
110	c
...	

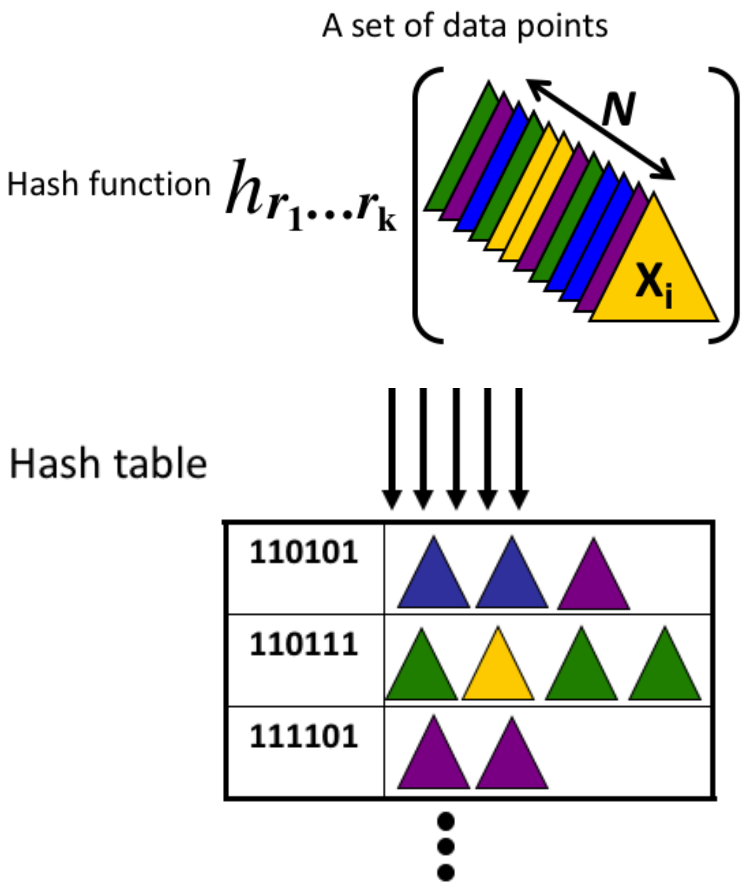
...	
001	d
010	b
011	a c e
...	

Table 2

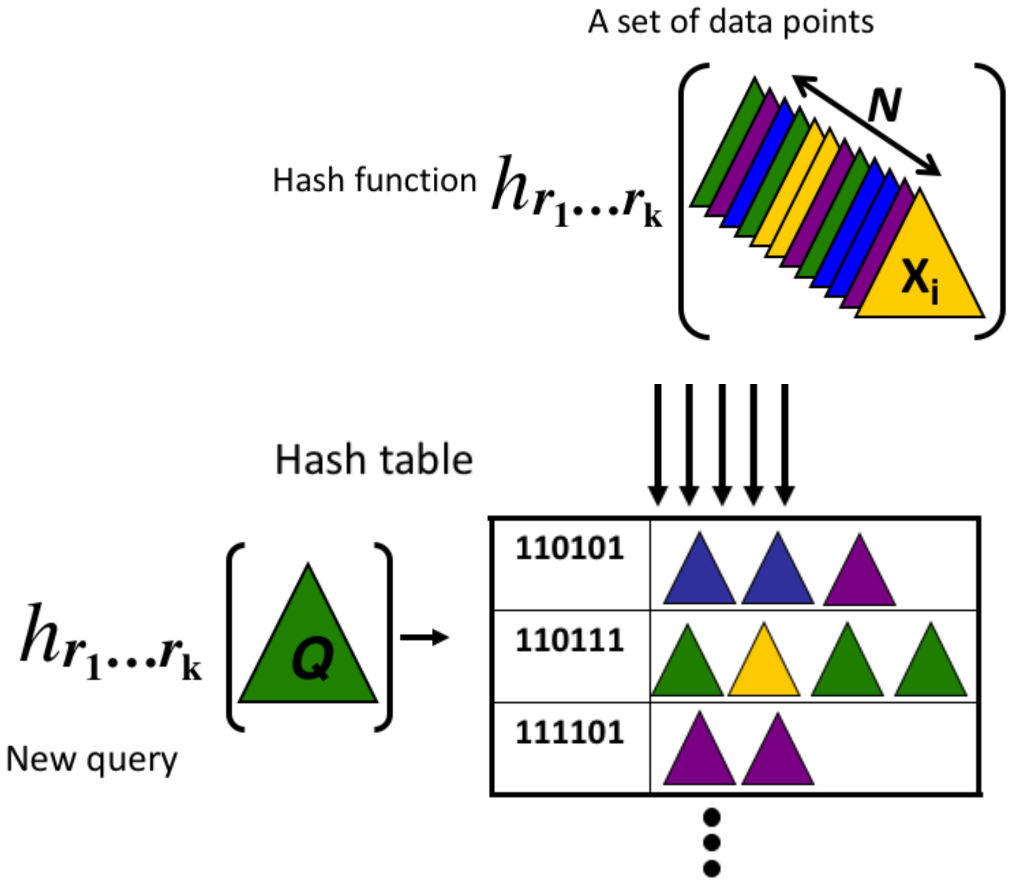
How to search from hash table?



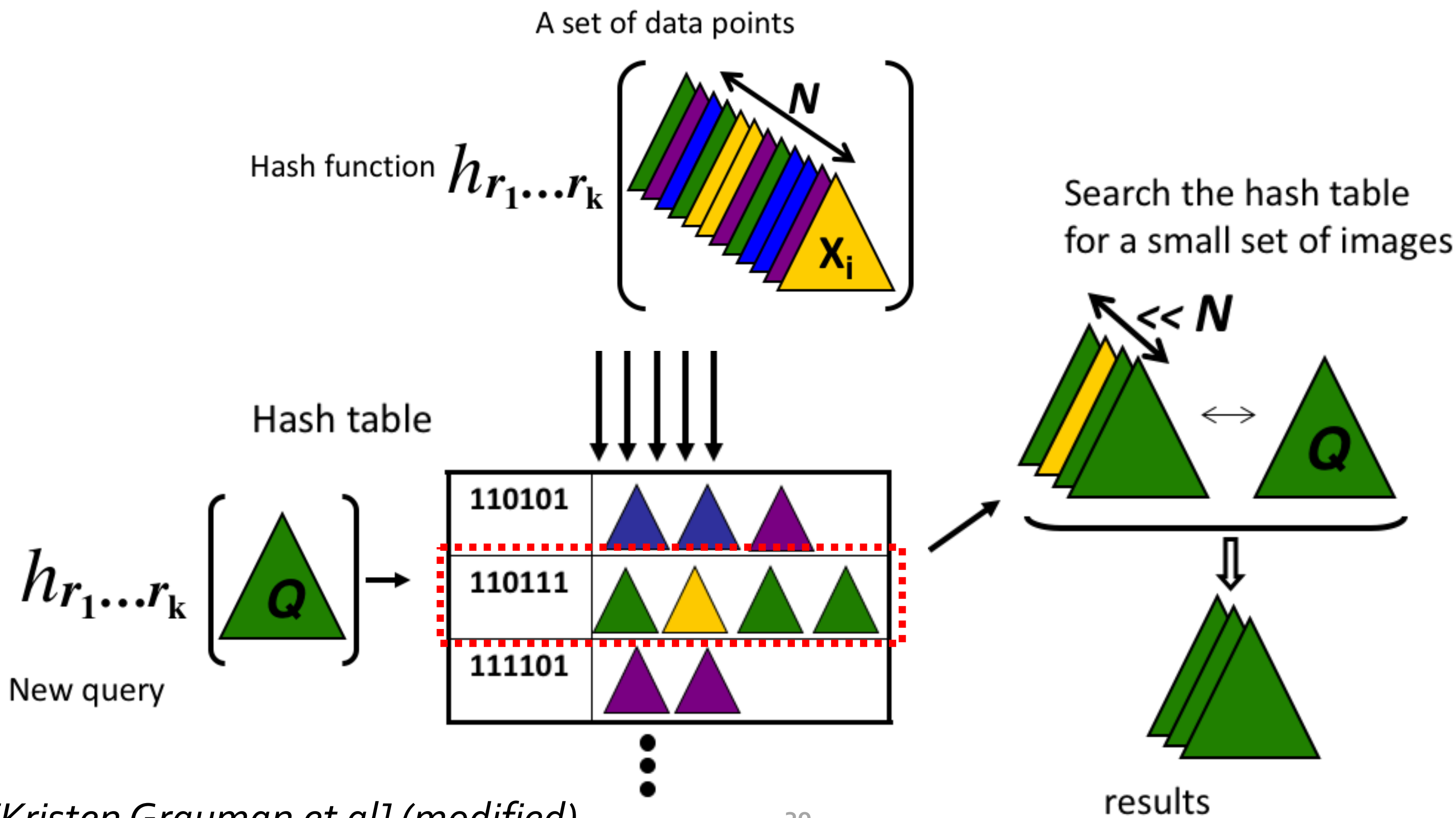
How to search from hash table?



How to search from hash table?



How to search from hash table?

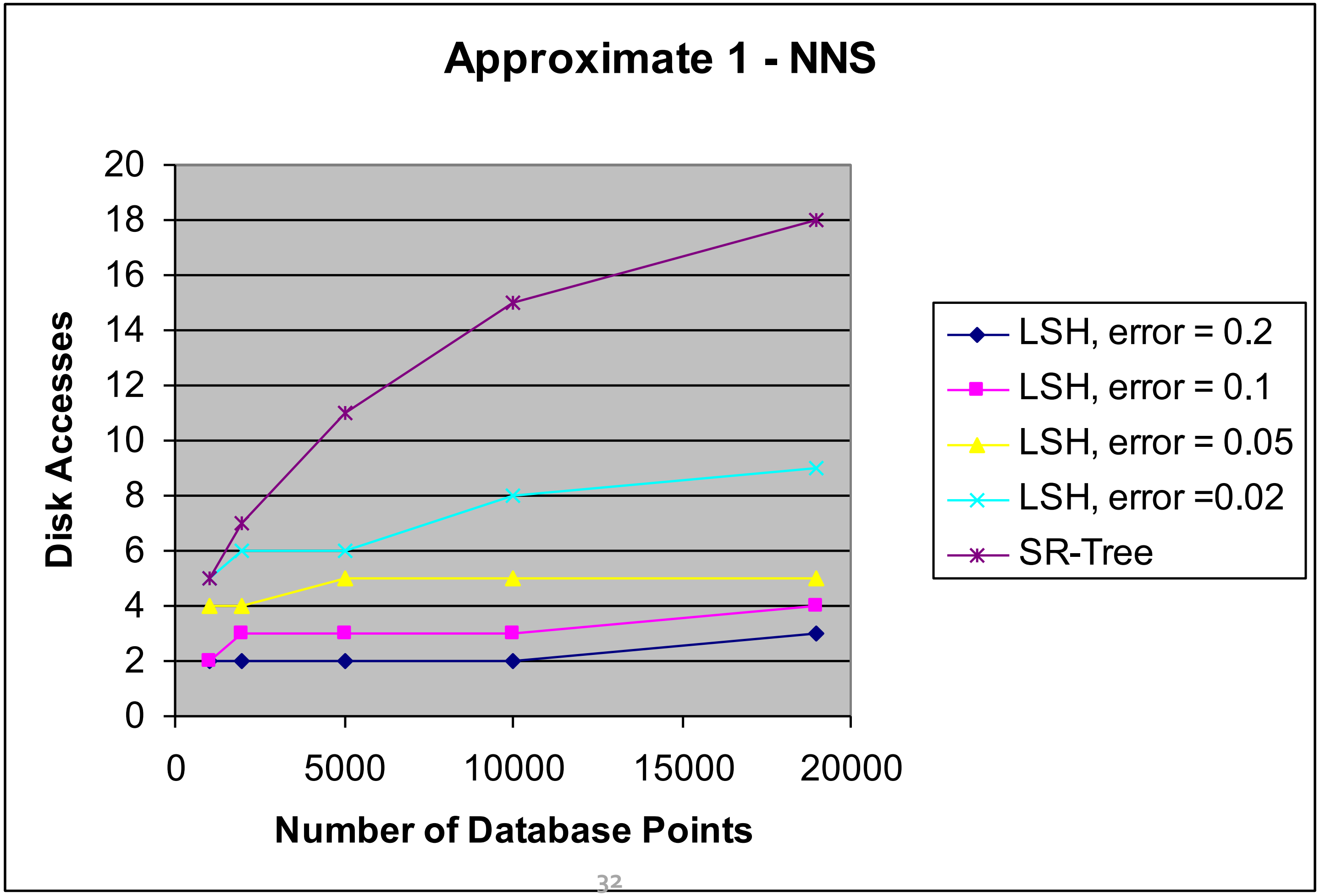


[Kristen Grauman et al] (modified)

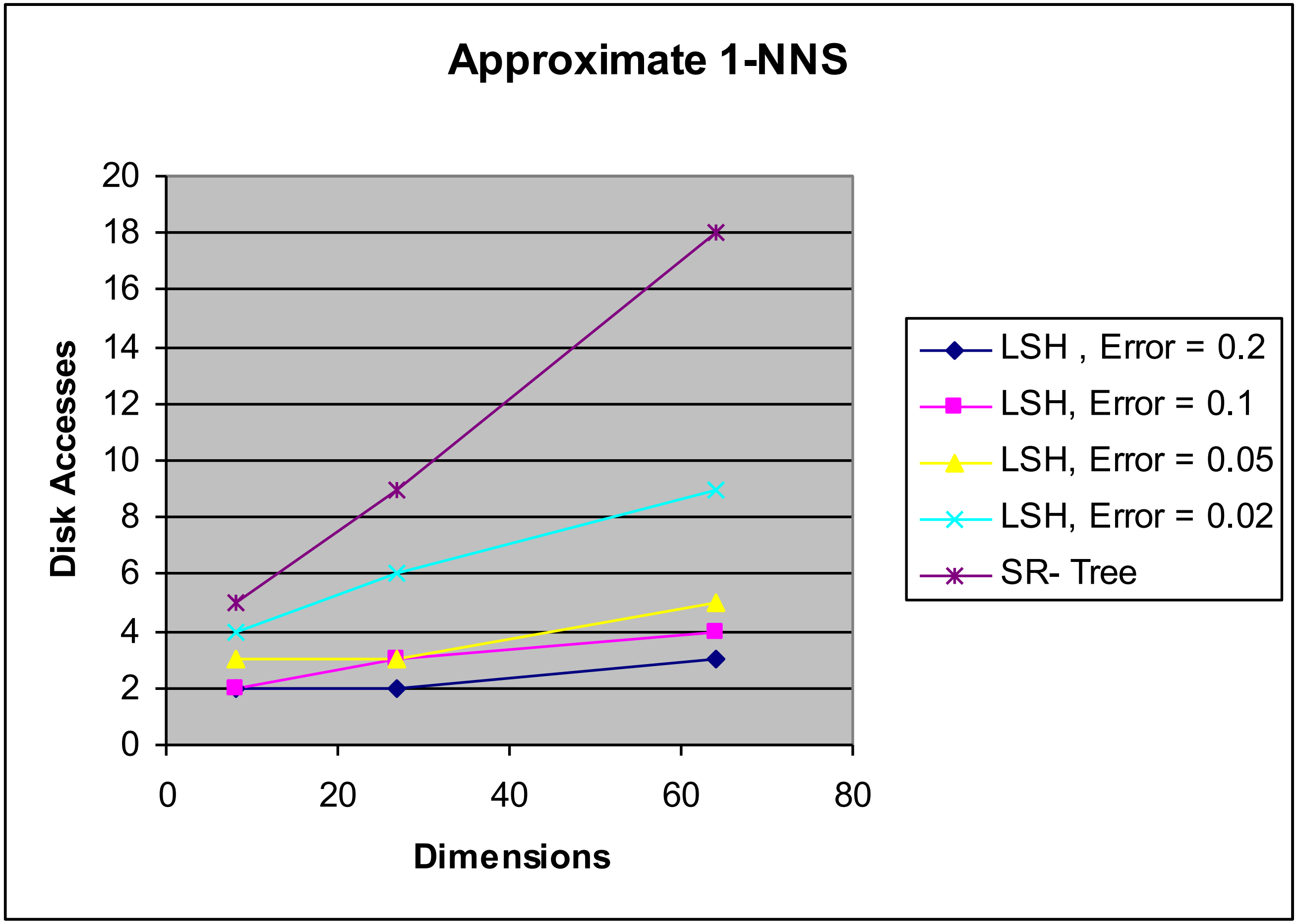
Performance

- Data Sets
  - Color images from COREL Draw library (20,000 points,dimensions up to 64)
  - Texture information of aerial photographs (270,000 points, dimensions 60)
- Evaluation
  - Speed, Miss Ratio, Error (%) for various data sizes, dimensions, and K values
  - Compare Performance with SR-Tree (Spatial Data Structure)

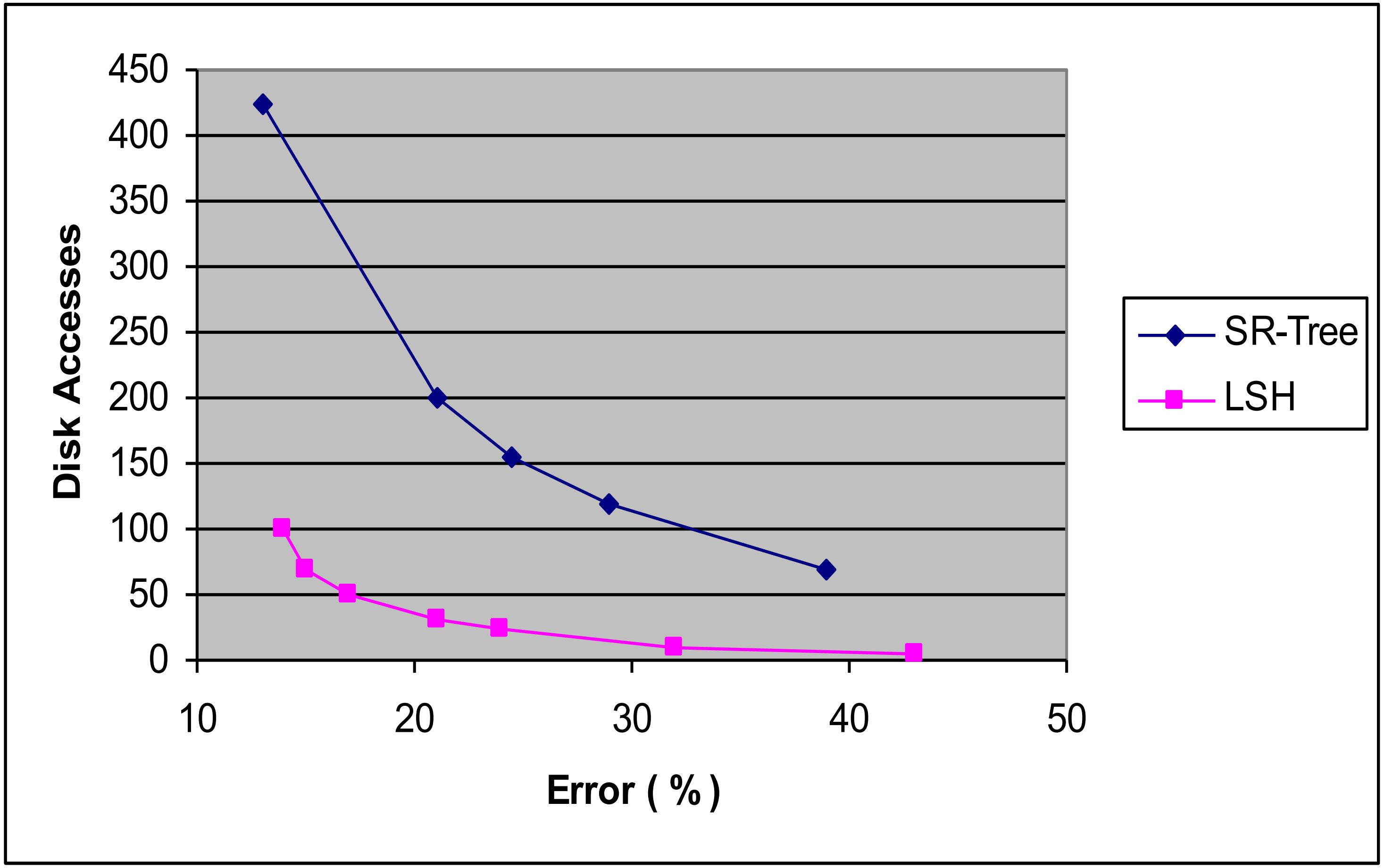
Speed vs. Data Size



Speed vs. Dimension



Speed vs. Error



Analysis

- LSH solves c-approximate NN with:
  - Number of hash functions: $L = \mathbf{n^{\rho}}$
  - $\boldsymbol{\rho} = \log(1/P_1)/\log(1/P_2)$
  - E.g., for the Hamming distance we have $\boldsymbol{\rho} = 1/c$
  - Constant success probability per query q
- Questions:
  - Can we extend this beyond Hamming distance?
  - Can we reduce the exponent $\boldsymbol{\rho}$?

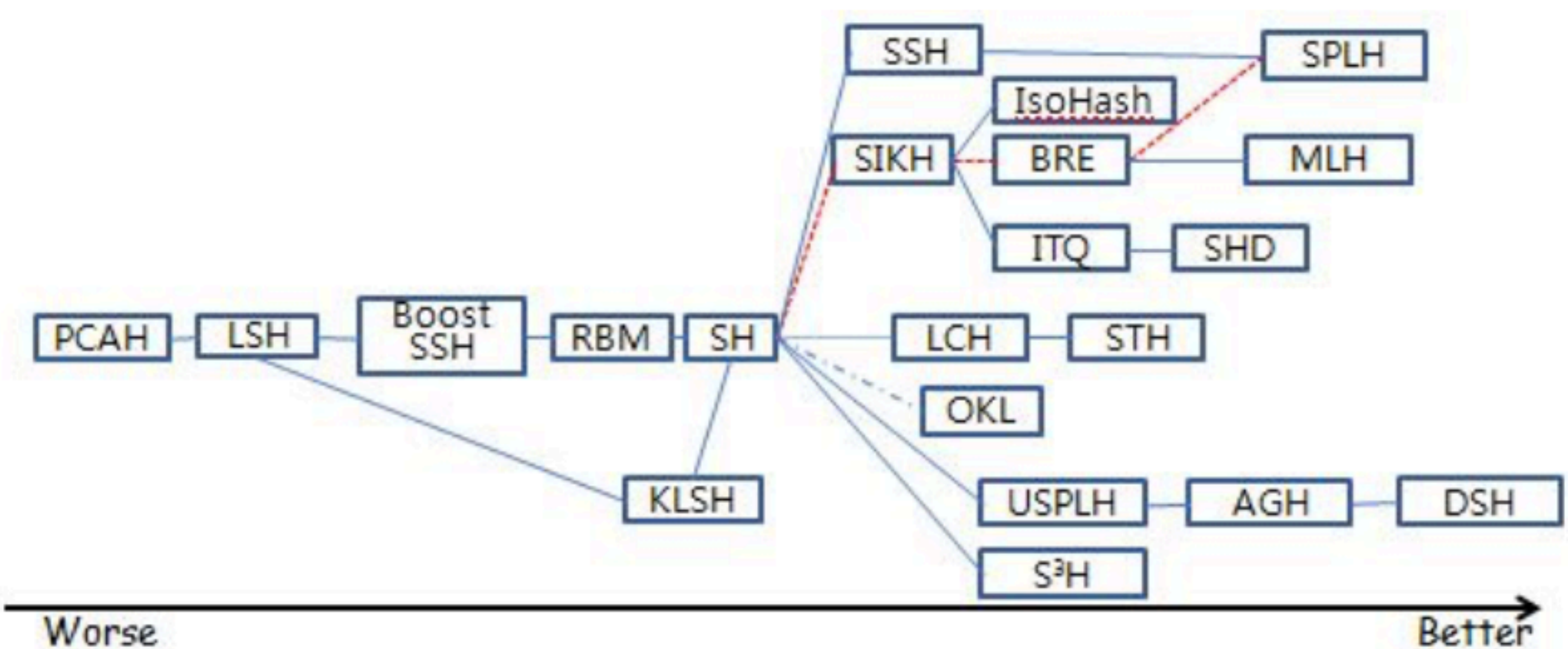
Analysis

Distance metric	$\rho = (\ln 1 / p_1) / (\ln 1 / p_2)$	c = 2	Reference
Euclidean (l_2)	$\leqq 1/c^2 + o(1)$	1/4	[Andoni, Indyk 2006]
	$\geqq 1/c^2 - o(1)$		[O'Donnell, Wu, Zhou 2011]
	$\dfrac{1}{2c^2 - 1} + o(1)$	1/7	[Andoni, R 2015]
Hamming (l_1)	$\leqq 1/c$	1/2	[Indyk, Motwani 1998]
	$\geqq 1/c - o(1)$		[O'Donnell, Wu, Zhou 2011]
	$\dfrac{1}{2c - 1} + o(1)$	1/3	[Andoni, R 2015]

Categorization of LSH approaches

- ✓ Number of bucket sets
- ✓ Shape of hash functions – (bit sampling, hyperplane, sine function, hypersphere)
- ✓ Data dependency
- ✓ Supervision
- ✓ Quantization - (single-bit quantization, multi-bits quantization)

Comparison [Lee, 2012]



Acknowledgments

PATREO
UFMG
30 ANOS FAPEMIG 1986 - 2016
CAPES
CNPq
Conselho Nacional de Desenvolvimento Científico e Tecnológico

References

Jin, J. S. (2003). Indexing and Retrieving High Dimensional Visual Features, pages 178–203. Springer Berlin Heidelberg, Berlin, Heidelberg.

Faloutsos, C., Barber, R., Flickner, M., Hafner, J., Niblack, W., Petkovic, D., and Equitz, W. (1994). Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3(3):231–262.

Idris, F. and Panchanathan, S. (1997). Review of image and video indexing techniques. *J. Vis. Comun. Image Represent.*, 8(2):146–166.

Wang, J., Shen, H. T., Song, J., and Ji, J. (2014). Hashing for similarity search: A survey. *CoRR*, abs/1408.2927.