Adaptive Binary Quantization for Fast Nearest Neighbor Search

Zhujin Li[1], Xianglong Liu[1*], Junjie Wu[1], and Hao Su[2]

[1]*Beihang University, Beijing, China*

[2]*Stanford University, Stanford, CA, USA*

http://www.nlsde.buaa.edu.cn/~xlliu

Outline

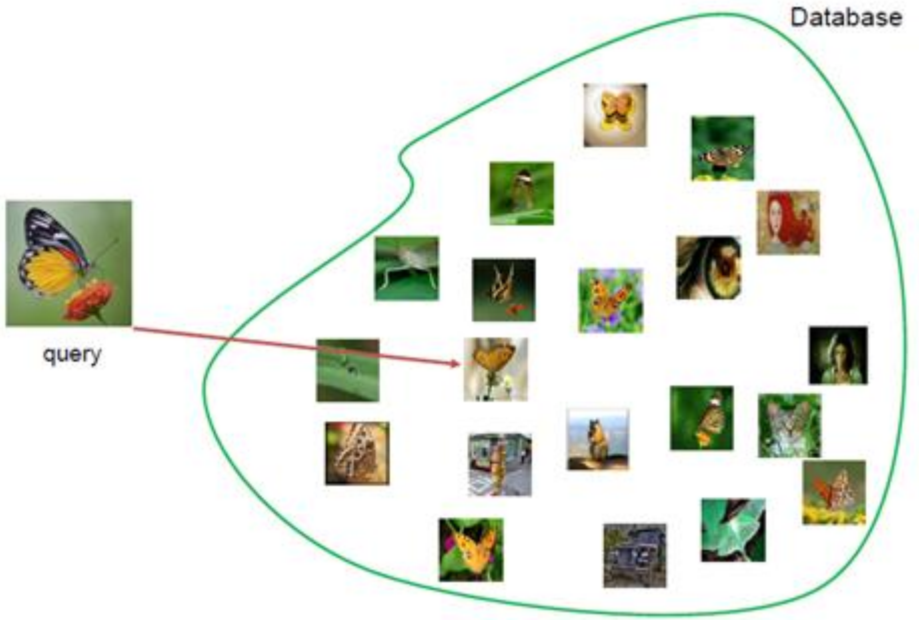
- **Introduction**
  - Nearest Neighbor Search
  - Motivation
- **Adaptive Binary Quantization**
  - Formulation
  - Optimization
- **Experiments**
- **Conclusion**

Introduction: Nearest Neighbor Search (1)

Definition

Given a database $P = \{p_i\}_{i=1 \dots n}$ and a query q, the nearest neighbor of q:

$$p^* \in P \text{ , such that } d(q, p^*) \leq d(q, p)$$



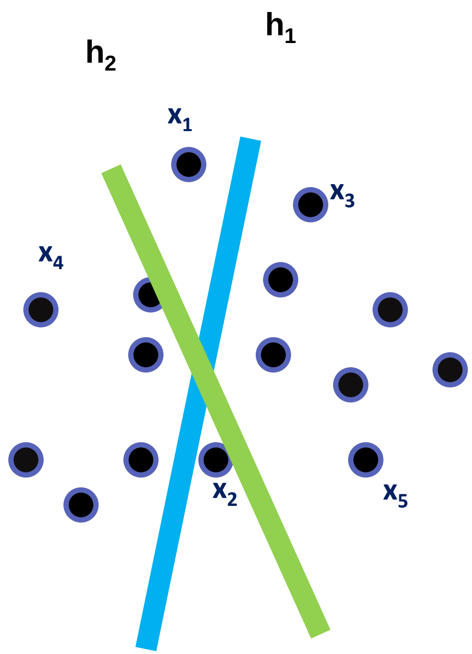
Solutions

- linear scan
  - time and memory consuming
- tree-based: KD-tree, VP-tree, etc.
  - divide and conquer
  - degenerate to linear scan for high dimensional data

Introduction: Nearest Neighbor Search (2)

- **Hash based nearest neighbor search**
  - Locality sensitive hashing [Indyk and Motwani, 1998]: close points in the original space have similar hash codes

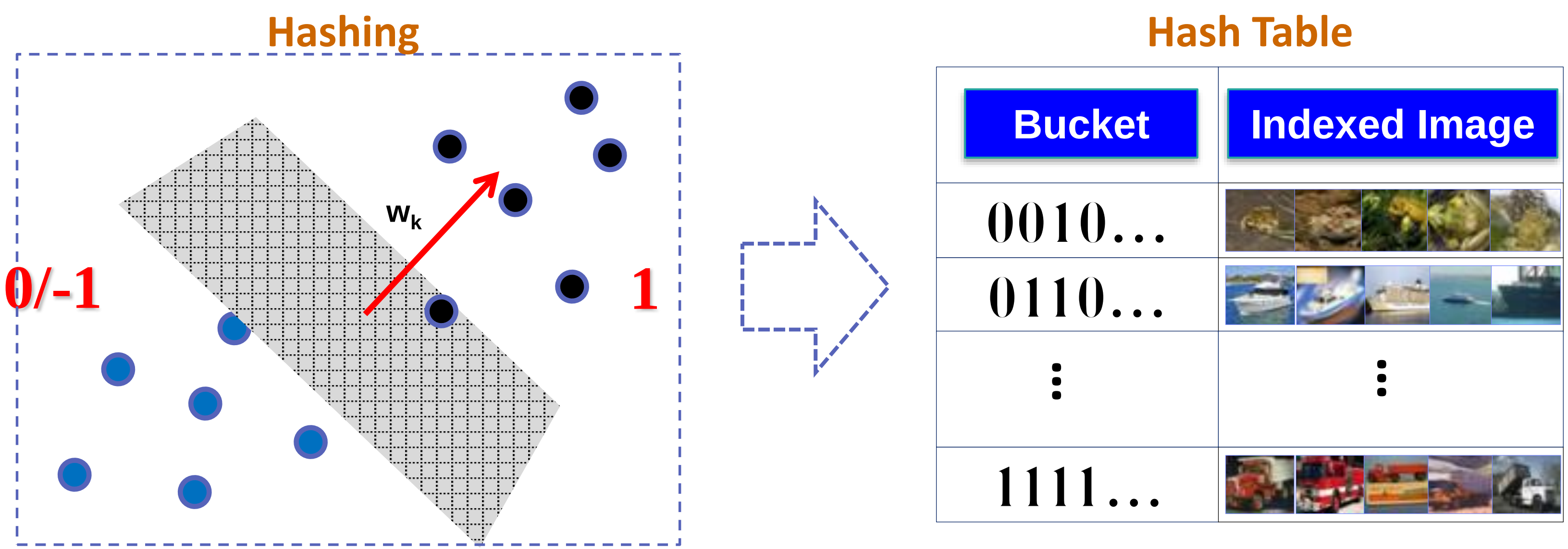
$$h(x) = sgn(w^T x + b)$$



X	x_1	x_2	x_3	x_4	x_5
h_1	0	1	1	0	1
h_2	1	0	1	0	1
…	…	…	…	…	…
h_k	…	…	…	…	…
	010…	100…	111…	001…	110…

Introduction: Nearest Neighbor Search (3)

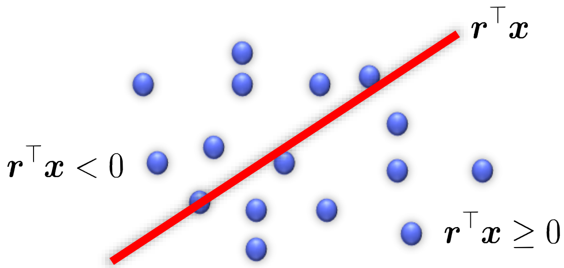
- **Hash based nearest neighbor search**
  - Compressed storage: binary codes
  - Efficient computations: hash table lookup or Hamming distance ranking based on binary operations



Introduction: State-of-the-art Hashing Solutions (1)

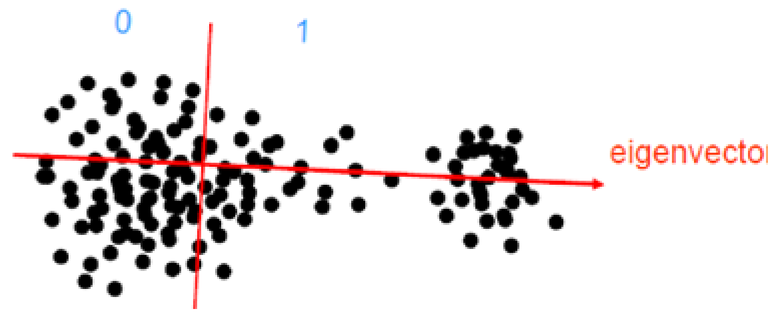
- **Linear projection based quantization**

LSH: random



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

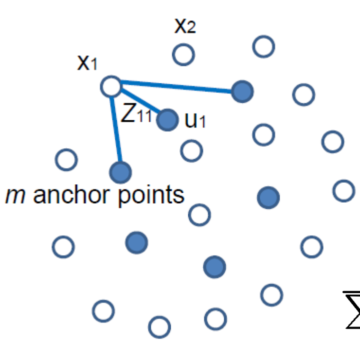
PCAH: PCA



$$h_k(x) = sgn(w_k^T x + b_k) \quad w_k \sim \text{eigenvec}(\text{Cov}(\mathbf{X}))$$

Try to capture the data distribution

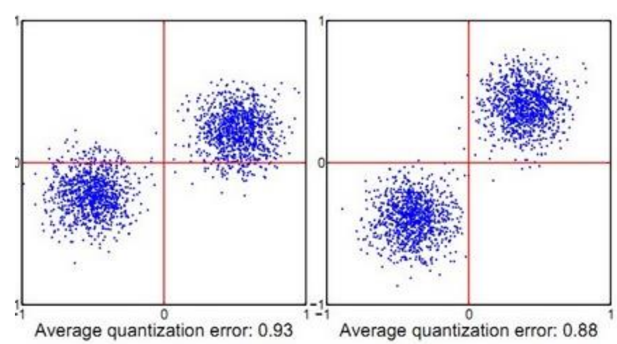
AGH: Kernel, ICML'11



$$\frac{\exp(-\mathcal{D}^2(x_i,u_j)/t)}{\sum_{j' \in \langle i \rangle} \exp(-\mathcal{D}^2(x_i,u_{j'})/t)}$$

$$h_k(x) = \operatorname{sgn}(\mathbf{z}^\top(\mathbf{x})\mathbf{a}_j$$

ITQ: Rotation, CVPR'12



(b) Random Rotation. (c) Optimized Rotation.

$$\min_{\mathbf{R}\mathbf{R}^\top=\mathbf{I}, \mathbf{B} \in \pm^{n \times r}} \|\mathbf{X}^\top \mathbf{W}\mathbf{R} - {}^{6}\mathbf{B}\|_F^2$$

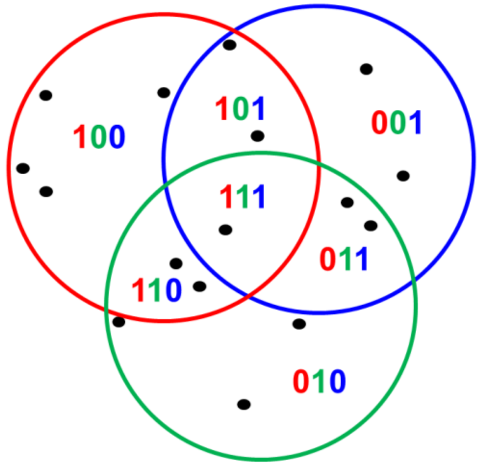
Introduction: State-of-the-art Hashing Solutions (2)

- **Prototype based quantization**
  - Step 1: find a number of prototypes to represent the data (like clustering)
  - Step 2: assign a binary code to the prototype

SPH, CVPR'12

each prototype generate a bit

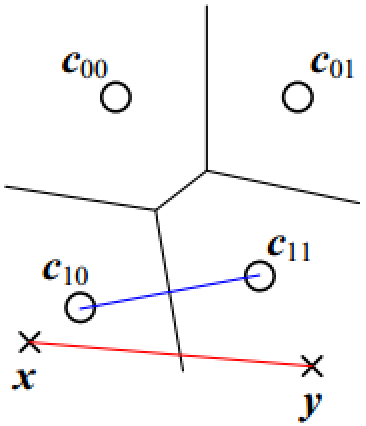
$\in \{\mathbf{0}, \mathbf{1}\}$ **with** 2 **codes**



KMH, CVPR'13

each prototype generate multiple bits

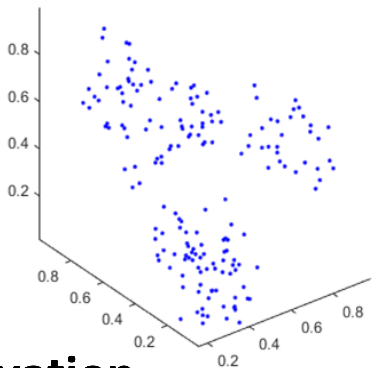
$\in \{\mathbf{0}, \mathbf{1}\}^m$ **with** 2^m **codes**



achieve encouraging performance

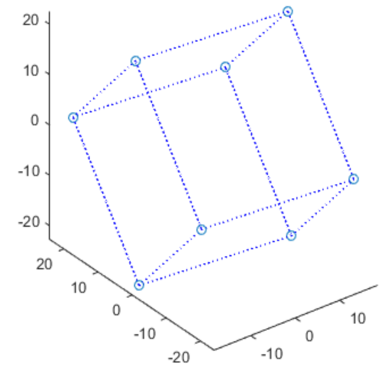
Introduction: Motivation

- **Problems**
  - make use of the complete binary code set (geometrically forms a hypercube), which can hardly characterize the real-world data distribution

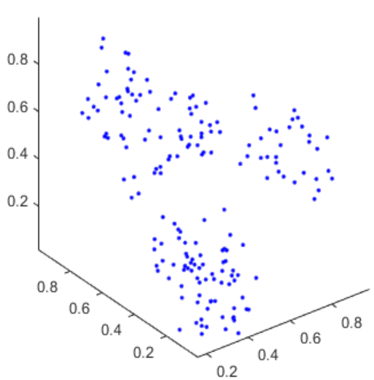


$$d_o(x,y) \cong d_h(c_x,c_y)$$

Using the complete binary code set

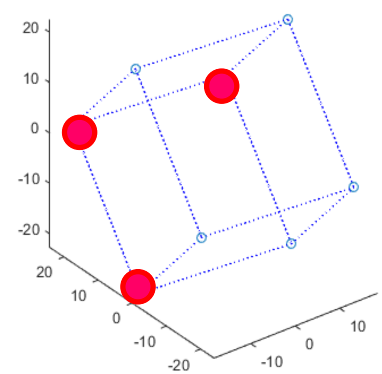


- **Motivation**
  - a better coding solution only relying on a small subset of binary codes (instead of the complete set) can largely reduce the quantization loss.



$$d_o(x,y) \cong d_h(c_x,c_y)$$

Using a small subset of binary codes



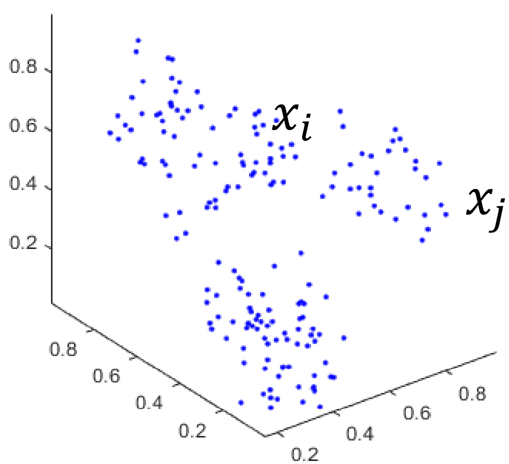
Outline

- **Introduction**
  - Nearest Neighbor Search
  - Motivation
- **Adaptive Binary Quantization**
  - Formulation
  - Optimization
- **Experiments**
- **Conclusion**

Adaptive Binary Quantization: Formulation (1)

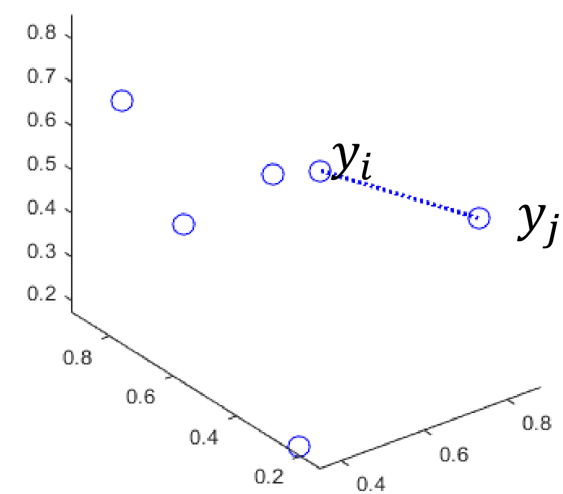
- **Goal:**
  - characterize the inherent data relations, and maintain the affinities between samples in the code space (i.e., Hamming space).

- **Basic idea: space alignment**
  - jointly find the discriminative prototypes and their associated binary codes that can align the Hamming space to the original one



the original space

$$d_o(x_i, x_j) \cong d_h(y_i, y_j)$$



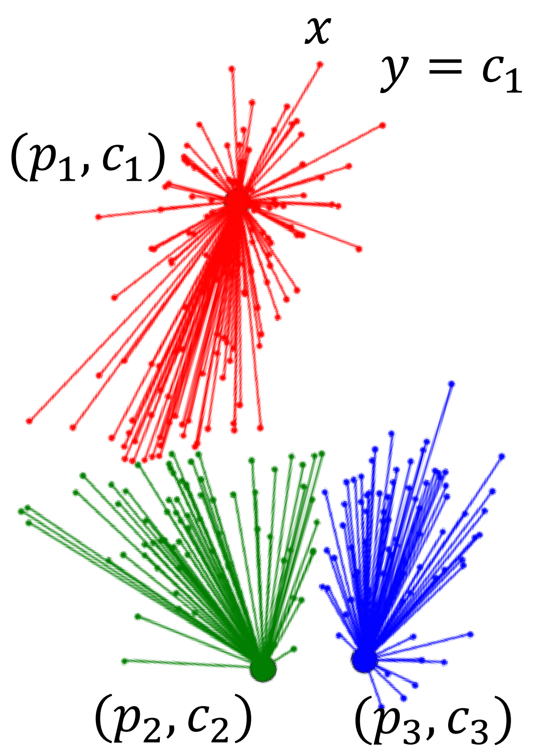
the Hamming space

Adaptive Binary Quantization: Formulation (2)

- **Notations**
  - The training data set $X = [x_1, x_2, \dots, x_N] \in \mathbb{R}^{d\times n}$
  - The code matrix $Y = [y_1, y_2, \dots, y_n] \in \{-1,1\}^{b\times n}$
  - The prototype set $P = \{p_k | p_k \in \mathbb{R}^{d\times n}\}$
  - The codebook $C = \{c_k | c_k \in \{-1, 1\}^b\}$

- **A prototype based hashing**
  - learn a hash function $h(x)$ that can map each x to y

$$h(x) = c_{i^*(x)} \quad i^*(x) = \arg\min_{k} d_o(x, p_k)$$



Adaptive Binary Quantization: Formulation (3)

- **Space Alignment**
  - concentrate on the distance consistence so that codes in Hamming space will be aligned with the original data distribution
    - global distribution: the prototypes capture the data distribution
    - neighbor structure: data belonging to the same prototype share the same code
  - Quantization loss

$$\boldsymbol{Q}(Y,X) = \frac{1}{n^2}\sum_{i,j=1}^{n}\left\|\lambda d_o(x_i,x_j) - d_h(y_i,y_j)\right\|^2$$

- $d_h(y_i,y_j) = \frac{1}{2}\|y_i - y_j\|$ is the square root of the Hamming distance

$$d_o(x_i,x_j) \approx d_o\left(x_i, p_{\mathrm{i}^*(\mathrm{x_j})}\right)$$

$$\boldsymbol{Q}\,(P,C,i^*(X)) = \sum_{i=1}^{n}\sum_{k=1}^{|P|}\frac{w_k}{n^2}\left\|\lambda d_o(x_i,x_j) - d_{h\left(c_{\mathrm{i}^*(\mathrm{x_j})_i},\, c_{\mathrm{j}^*(\mathrm{x_j})}\right)}\right\|^2$$

Adaptive Binary Quantization: Optimization (1)

- **Space Alignment**

$$\min_{P,C,i^*(X)} \boldsymbol{Q}\left(P, C, i^*(X)\right)$$
$$s.t.\quad c_k \in \{-1,1\}^b\ ; \qquad c_k^T c_l \neq b,\ \ l \neq k$$

- **Alternating Optimization**
  - **1. Adaptive Coding**

  fixing P and $i^*(X)$, optimize C

  - **2. Prototype Update**

  fixing C and $i^*(X)$, optimize P

  - **3. Distribution Update**

  fixing P and C, optimize $i^*(X)$

Algorithm 1 Adaptive Binary Quantization.

Input: Training data $\mathbf{X}$, and the binary code length b.

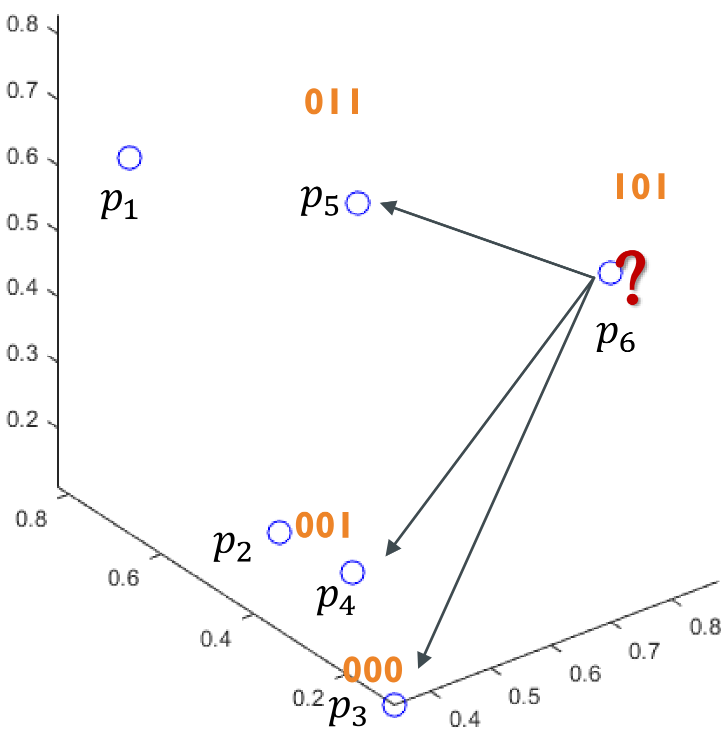
Output: Hash function h, the prototype set $\mathcal{P}$ and the corresponding binary code set $\mathcal{C}$.

Initialize the assignment index $i^*(\mathbf{X})$ and the prototype set $\mathcal{P}$ using k-means.
Initialize the scale parameter λ according to (11).
repeat
    for $l = 1, \ldots, |\mathcal{P}|$ **do**
        Find the local optimal code $\mathbf{c}_l$ for $\mathbf{p}_l$ by solving (6);
    end for
    Update the prototype set $\mathcal{P}$ according to (8) and (9);
    Update the distribution $i^*(\mathbf{X})$ according to (10);
until convergence

Adaptive Binary Quantization: Optimization (2)

- **Adaptive Coding**
  - With the prototype P and the assignment index $i^*(X)$, from 2^b codes find a subset most consistent with the prototypes.

$$\min_{c_k \in \tilde{C}} \sum_{i^*(x_i)=k} \sum_{k' \neq k} w_{k'} \|\lambda d_o(x_i, p_{k'}) - d_h(c_k, c_{k'})\|^2 + \sum_{i^*(x_i) \neq k} w_k \left\|\lambda d_o(x_i, p_k) - d_h\left(c_{i^*(x)}, c_k\right)\right\|^2$$



Codebook

~~011~~
~~001~~
~~000~~
010
✓ 100
101
110
111

Adaptive Binary Quantization: Optimization (3)

- **Prototype Update**
  - With the codebook C and the assignment index $i^*(X)$, find the prototypes P that can simultaneously capture the data distribution and align with the geometric structure in the code space

$$\min_{k' \leq |C|} \sum_{k=1}^{|C|} w_k \| \lambda d_o(x_i, p_k) - d_h(c_{k'}, c_k) \|^2 \quad \Rightarrow \quad p_k = \frac{1}{w_k} \sum_{i^*(x_i)=k} x_i$$

  - prototypes P might be shrunk, and thus gradually adapt the binary codes to the data distribution

- **Distribution Update**
  - an assignment updating step to capture the distribution variation

$$i^*(x_i) = \arg\min_{k \leq |P|} d_o(x_i, p_k)$$

Adaptive Binary Quantization: Algorithm Details (1)

- **Initialization**
  - k-means clustering to initialize the prototypes P
  - 2^b prototypes and codes to initialize scale parameter λ

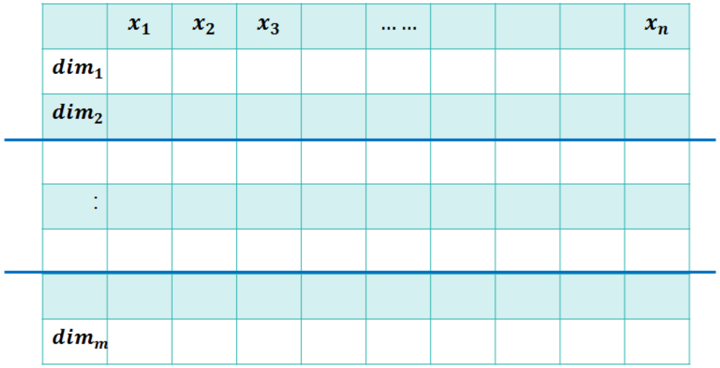
$$\lambda = \frac{\frac{1}{2^b}\sum_{\mathbf{c}_k,\mathbf{c}_l\in\{-1,1\}^b} d_h(\mathbf{c}_k,\mathbf{c}_l)}{\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{2^b} d_o(\mathbf{x}_i,\mathbf{p}_k)}$$

- **Product Quantization**
  - Generating long (b^*) hash codes by

(1) dividing the original space into $M = b^*/b$ subspaces

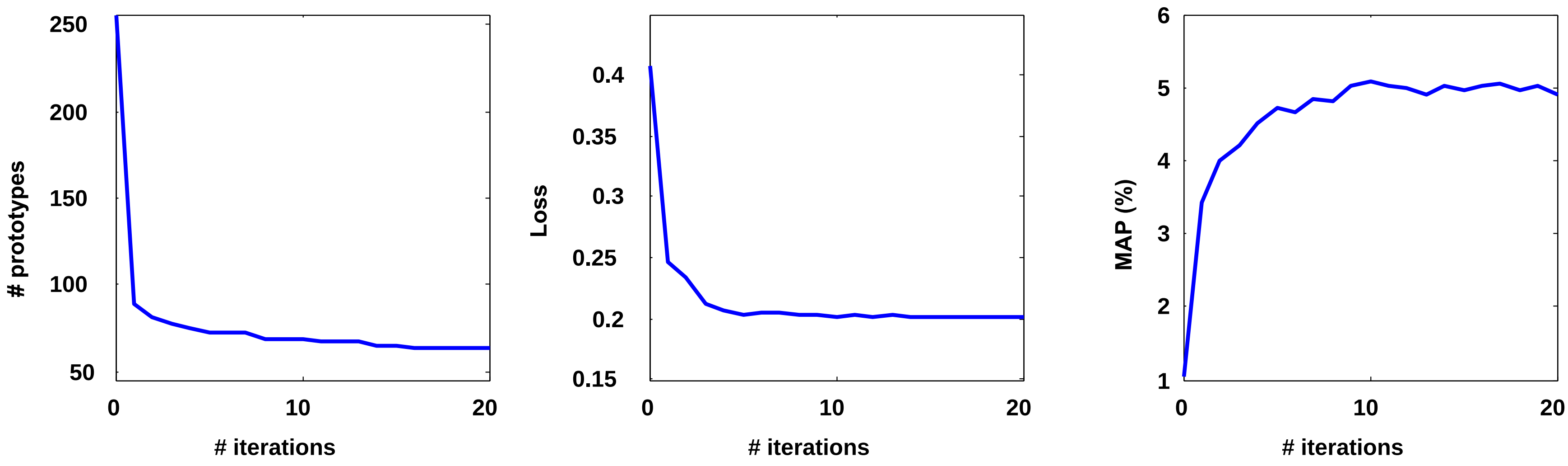
(2) adaptive binary quantization in each subspace

$$d_o(\mathbf{x}_i,\mathbf{x}_j) \approx d_o(\mathbf{x}_i,\mathbf{p}_{i^*(\mathbf{x}_j)}) = \sqrt{\sum_{m=1}^{M} d_o(\hat{\mathbf{x}}_i^{(m)}, \hat{\mathbf{p}}^{(m)}_{i^*(\hat{\mathbf{x}}_j^{(m)})})^2}$$



Adaptive Binary Quantization: Algorithm Details (2)

- **Complexity**
  - Training: for n training samples of dimension d, to generate b^* binary codes, the complexity $O(2^{2b}t \cdot nd)$, almost linear to n ($b \leq 8$ and # iteration $t \leq 20$)
  - Testing: $O(|P|d)$, close to the linear projection based hashing



Experiments

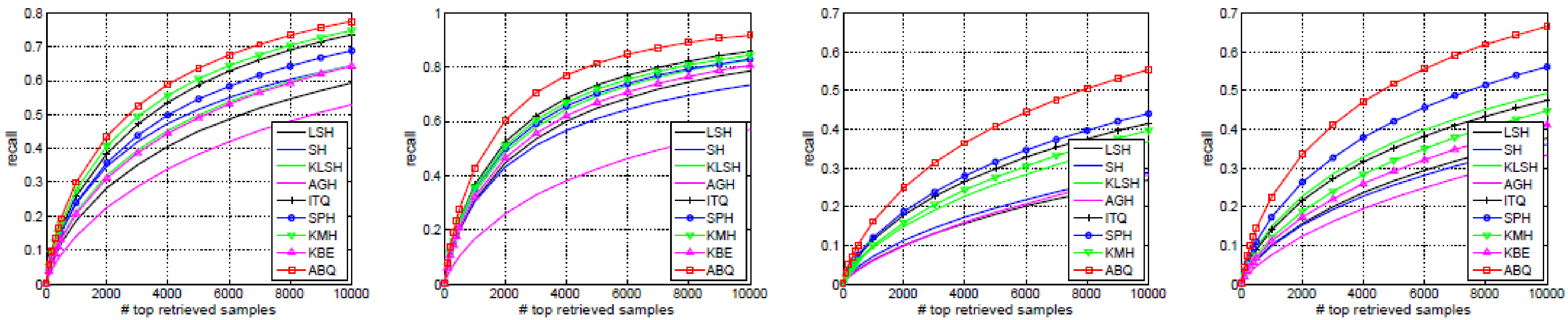
- **Datasets**
  - SIFT-1M: 1 million 128-D SIFT; GIST-1M: 1 million 960-D GIST
  - SIFT-20M: 20 millions 128-D SIFT; Tiny-80M: 80 millions 384-D GIST
- **Baselines:**
  - Projection based: LSH, SH, KLSH, AGH, ITQ, KBE
  - Prototype based: SPH, KMH
- **Setting:**
  - 50,000 and 100,000 training samples and 3,000 queries on each set
  - The groundtruth for each query is defined as the top 1,000 nearest neighbors on SIFT-1M, GIST-1M and SIFT-20M, and 5,000 on Tiny-80M based on Euclidean distances
  - Average performance of 10 independent runs

Experiments: precision performance

		MAP			PH (32 BITS)		TIME (128 BITS)	
		32 BITS	64 BITS	128 BITS	$r=1$	$r=2$	TRAIN (S)	SEARCH (S)
SIFT-1M	LSH	5.43±0.30	13.00±0.82	26.04±0.68	18.89	19.70	0.03	0.02
	SH	10.70±0.58	17.84±0.37	25.30±0.59	32.20	41.93	0.25	0.25
	KLSH	7.08±0.44	15.61±0.57	29.48±0.72	23.72	23.32	0.28	0.02
	AGH	6.26±0.27	9.11±0.31	11.10±0.23	15.90	11.93	0.55	0.04
	ITQ	9.70±0.14	20.14±0.47	33.23±0.49	28.38	22.09	5.08	0.16
	SPH	8.57±0.12	18.23±0.54	31.11±0.14	26.90	30.82	8.93	0.04
	KMH	11.51±0.27	22.50±0.31	32.06±0.52	35.63	40.00	680.64	0.12
	KBE	6.43±0.31	14.73±0.61	27.65±0.57	20.62	16.97	3.28	0.02
	ABQ	**12.47**±0.26	**24.92**±0.61	**41.34**±0.56	**41.30**	**43.09**	40.37	0.06
GIST-1M	LSH	1.34±0.08	3.15±0.07	5.97±0.19	5.41	7.15	0.21	0.05
	SH	1.90±0.23	3.19±0.19	4.92±0.19	8.94	6.58	1.70	0.24
	KLSH	2.41±0.09	5.23±0.18	9.76±0.23	9.31	10.70	0.44	0.05
	AGH	2.09±0.15	3.05±0.10	3.98±0.14	5.55	4.13	0.90	0.09
	ITQ	4.43±0.06	6.93±0.10	9.49±0.15	14.08	17.8	5.87	0.17
	SPH	3.65±0.14	6.97±0.10	11.52±0.19	12.20	17.05	25.24	0.07
	KMH	3.58±0.18	5.57±0.07	6.92±0.07	14.77	17.39	2380.61	0.15
	KBE	-	-	6.58±0.22	-	-	13.66	0.06
	ABQ	**4.92**±0.06	**10.06**±0.20	**16.10**±0.17	**23.46**	**17.84**	46.10	0.10

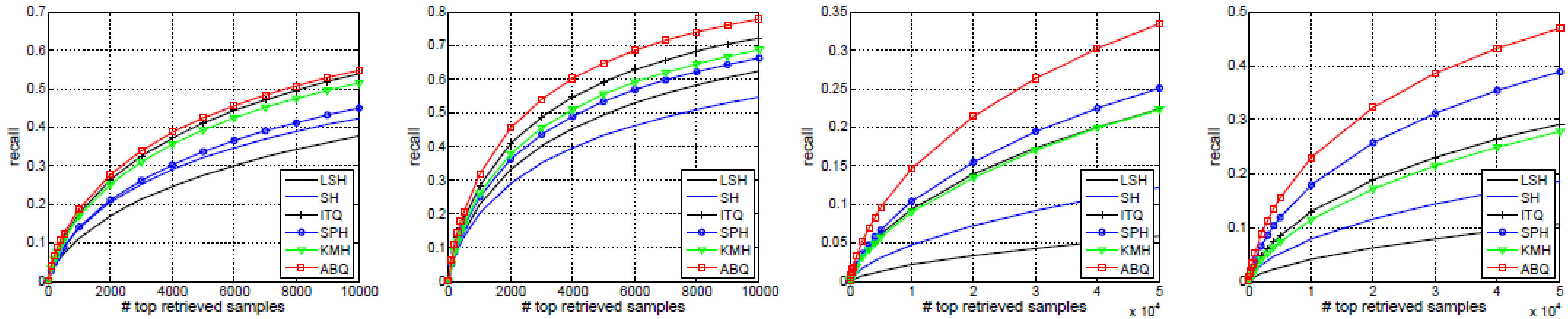
	SIFT-20M					TINY-80M				
	P@1,000			PH (32 BITS)		P@1,000			PH (32 BITS)	
	32 BITS	64 BITS	128 BITS	$r=1$	$r=2$	32 BITS	64 BITS	128 BITS	$r=1$	$r=2$
LSH	4.34	11.26	22.81	9.86	8.03	0.75	2.18	4.24	0.83	0.51
SH	8.00	13.91	20.19	22.70	17.34	2.77	5.12	9.06	3.37	1.71
ITQ	8.48	17.69	28.47	20.18	14.69	5.25	9.99	14.10	10.59	7.47
SPH	6.06	14.06	25.09	14.72	10.10	4.53	10.90	20.03	9.51	5.67
KMH	8.29	16.90	26.08	22.54	16.18	5.31	9.41	11.92	11.64	7.50
ABQ	**8.95**	**18.92**	**31.86**	**25.97**	**17.59**	**7.51**	**15.93**	**26.56**	**12.67**	**7.57**

Experiments: recall performance



(a) 64 bits on SIFT-1M (b) 128 bits on SIFT-1M (c) 64 bits on GIST-1M (d) 128 bits on GIST-1M

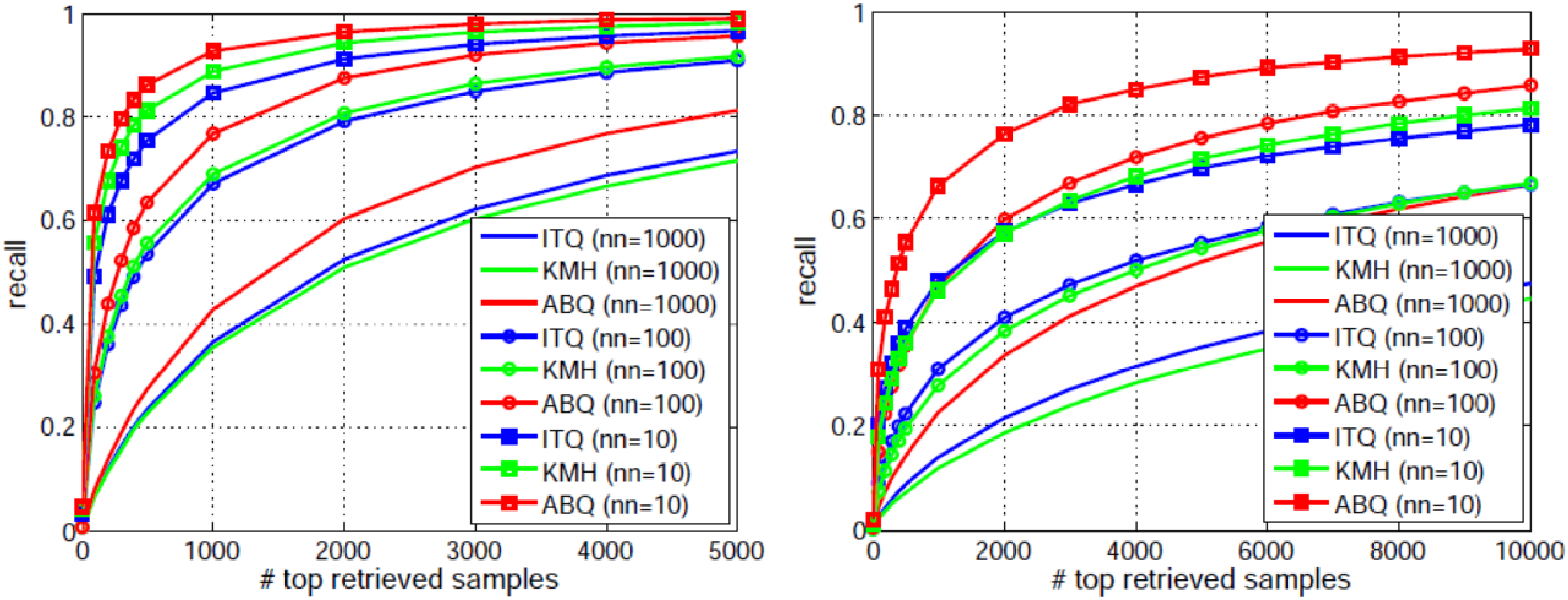
Figure 3: Recall performance of different hashing methods on SIFT-1M and GIST-1M.



(a) 64 bits on SIFT-20M (b) 128 bits on SIFT-20M (c) 64 bits on Tiny-80M (d) 128 bits on Tiny-80M

Figure 4: Recall performance of different hashing methods on SIFT-20M and Tiny-80M.

Experiments: effect of #groundtruth



(a) recall on SIFT-1M

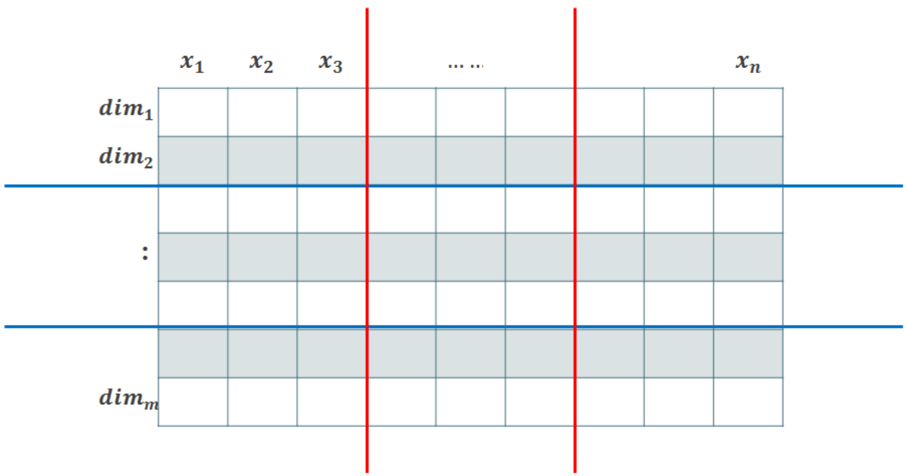
(b) recall on GIST-1M

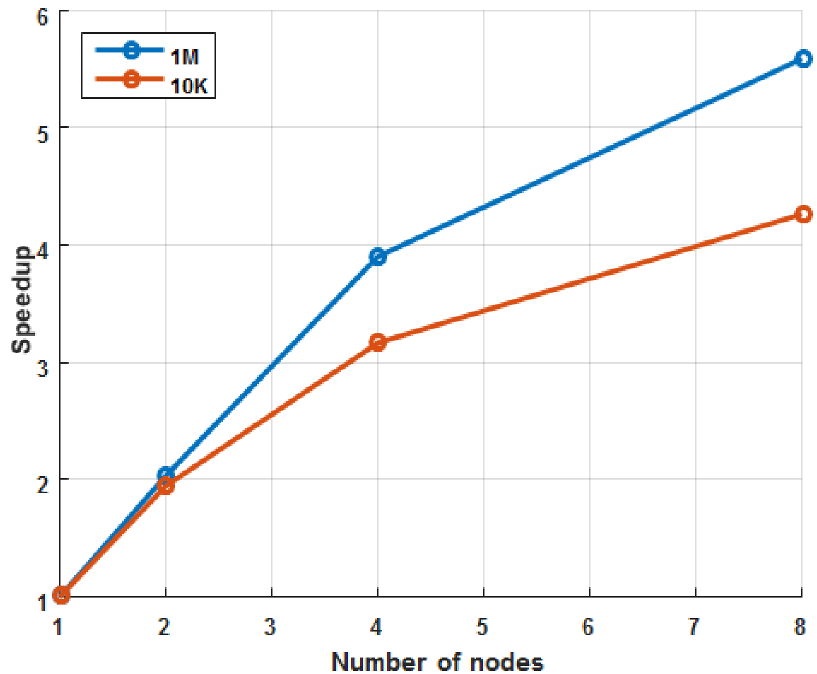
Conclusion

- **One observation:** in prototype based hashing there might exist a better coding solution that only utilizes a small subset of binary codes instead of the complete set
- **An adaptive binary quantization method**: jointly pursues a set **of** prototypes in the original space and a subset of binary codes in the Hamming space.
- **Good properties**: enjoys fast computation and the capability of generating long hash codes in product space, with discriminative power for nearest neighbor search.
- **Encouraging performance:** significantly outperforms existing methods on several large datasets, encouraging the further study on the effective binary quantization

Future work

- **Easy to extend for distributed system**
  - Distributed (map-reduce) + Parallel (PQ)





Thank you!

http://www.nlsde.buaa.edu.cn/~xlliu