Spherical hashing

Aleksey Morozov

Moscow Institute of Physics and Technology

November 23, 2017

Plan

- Introduction
- Releated work
- Spherical Hashing
- Evaluation
- Conclusion

Introduction I

- Huge image databases pose a significant challenge in terms of scalability to many computer vision applications, especially those applications that require efficient similarity search.
- Encoding high-dimensional data points into binary codes based on hashing techniques enables higher scalability thanks to both its compact data representation and efficient indexing mechanism.

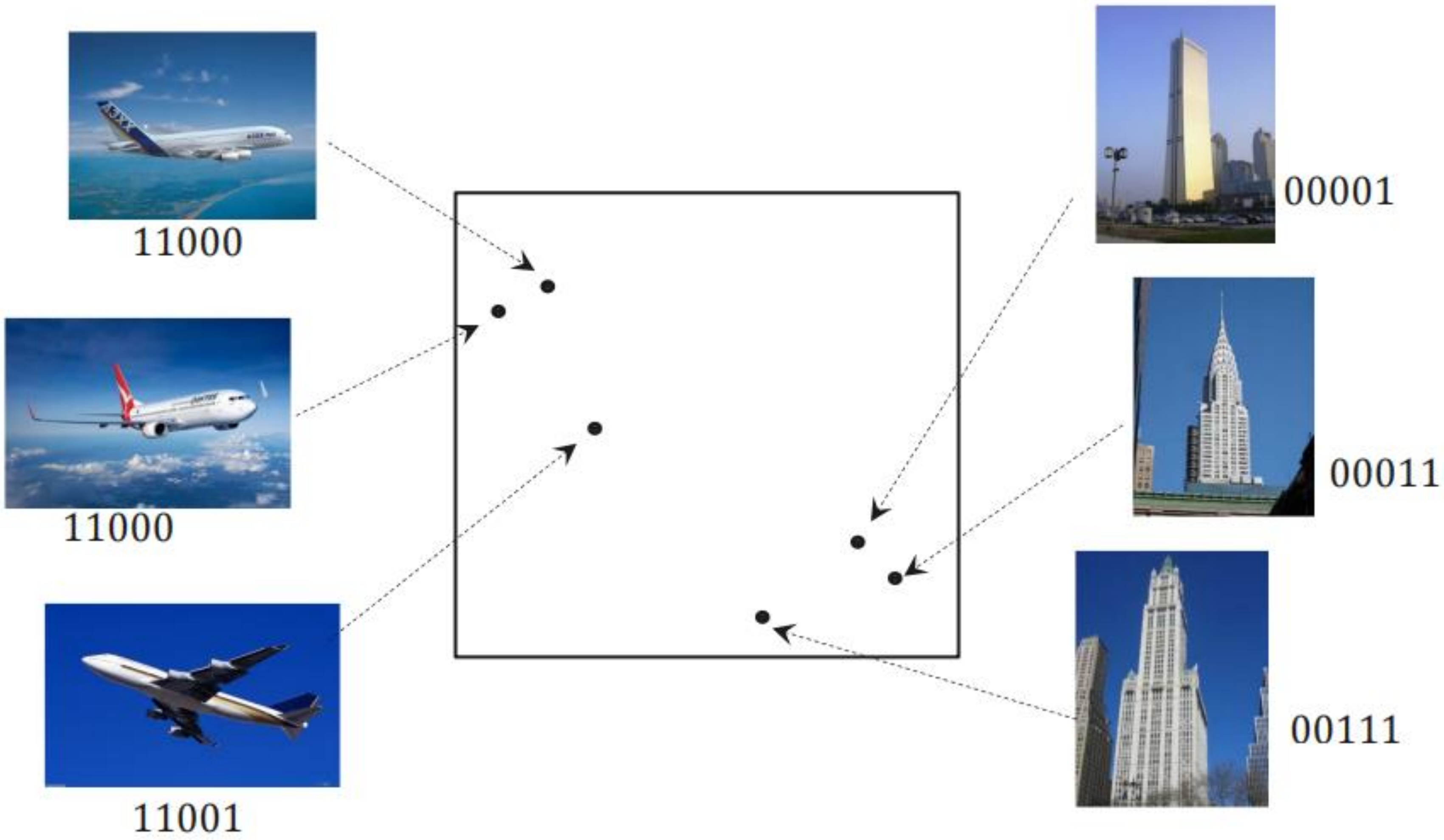
Introduction II

- Existing hashing techniques can be broadly categorized as
  - data-independent: hashing functions are chosen independently from the input points.
  - data-dependent: developing data-dependent techniques to consider the distribution of data points and design better hashing functions.
- In all of these existing hashing techniques, hyperplanes are used to partition the data points into two sets and assign two different binary codes (e.g., 0 or +1) depending on which set each point is assigned to.

Problem I

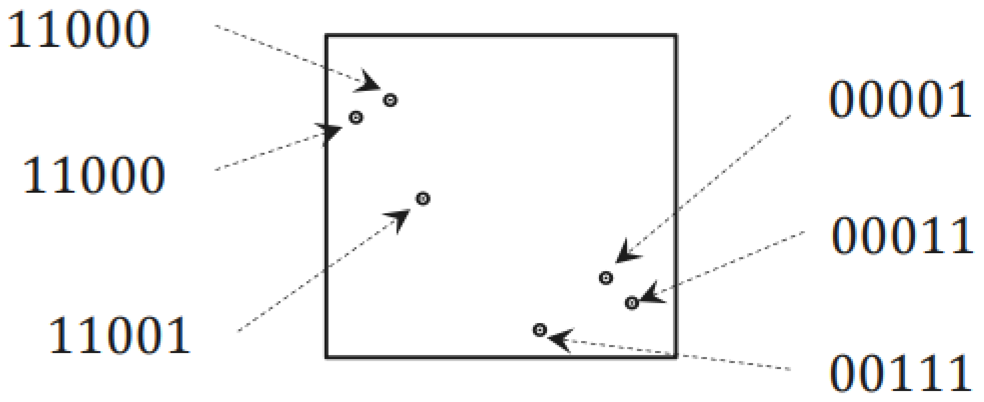
- Approximate k-nearest neighbor search in high dimensional space:
  - widely used in various applications
  - high computation cost, memory requirement
  - tree-based methods do not give any benefit (curse of dimensionality)
  - spatial hashing techniques get more attention

Binary Codes



From http://sglab.kaist.ac.kr/Spherical_Hashing/

Binary Codes



- Benefits:
  - high compression ratio (scalability)
  - fast similarity calculation with Hamming distance (efficiency)
- Issue:
  - how well do binary codes preserve data positions and their distances (accuracy)

From http://sglab.kaist.ac.kr/Spherical_Hashing/

Motivation

- Propose a novel spherical hashing scheme, analyze its ability in terms of similarity search, and compare it against the state-of-the-art hyperplane-based techniques
- Develop a new binary distance function tailored for the spherical hashing method.
- Formulate an optimization problem that achieves both balanced partitioning for each hashing function and the independence between any two hashing functions. Also, an efficient, iterative process is proposed to construct spherical hashing functions

Related work

State-of-the-art Methods

- Random hyper-planes from a specific distribution [Indyk – STOC 1998, Raginsky – NIPS 2009]
- Spectral graph partitioning [Weiss – NIPS 2008]
- Minimizing quantization error (ITQ) [Gong – CVPR 2011]
- Independent component analysis (ICA) [He – CVPR 2011]
- Support vector machine (SVM) [Joly – CVPR 2011]

All of them use hyperplanes.

Spherical Hashing

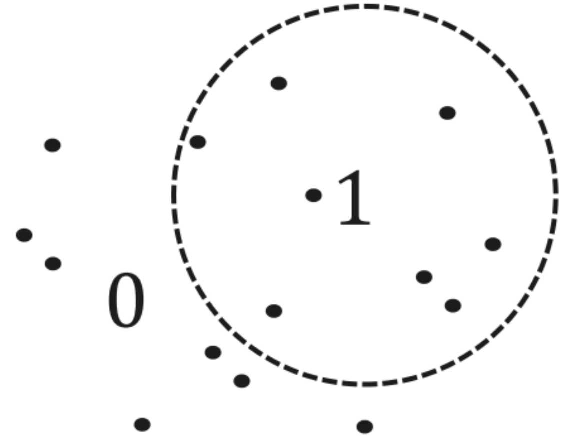
Set of n data points in a D-dimensional space $X = \{x_1, \dots, x_n\}$, $x_i \in \mathbb{R}^D$.

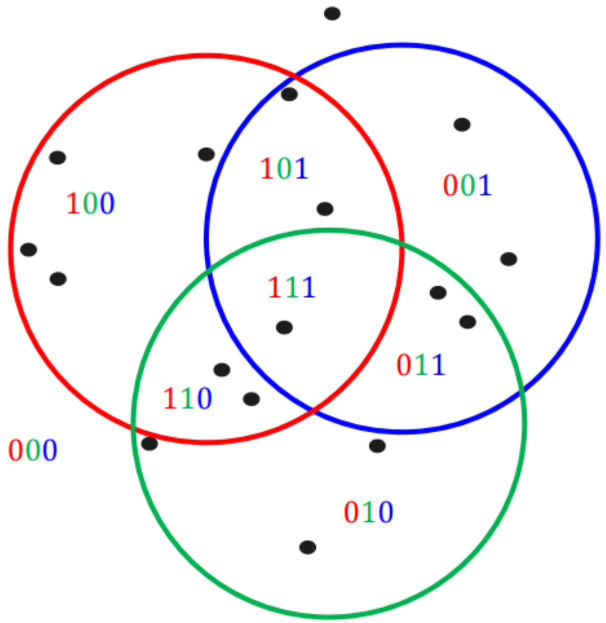
A binary code corresponding to each data point x_i is defined by $b_i = \{0, +1\}^c$, where c is the length of the code.

Hashing function $H(x) = \left(h_1(x), \dots, h_c(x)\right)$ maps points in $\mathbb{R}^D$ into the binary cube $\{0, +1\}^c$.

Each spherical hashing function $h_k(x)$ is defined by a pivot $p_k \in \mathbb{R}^D$ and a distance threshold $t_k \in \mathbb{R}^+$:

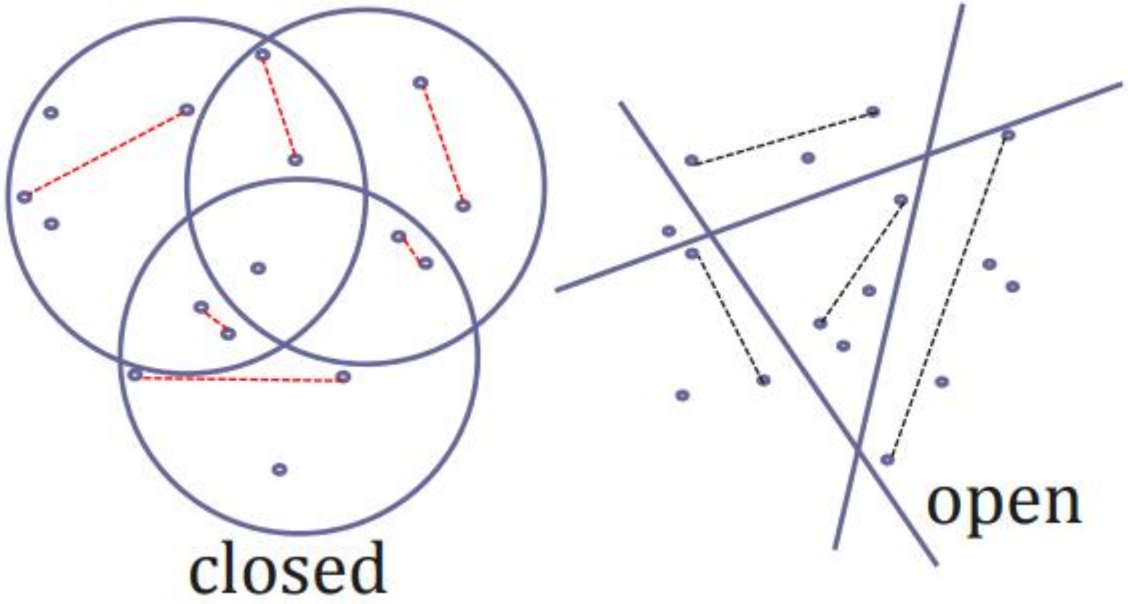
$$h_k(x) = \begin{cases} 0 \; when \; d(p_k, x) > t_k \\ +1 \; when \; d(p_k, x) \le t_k \end{cases}.$$

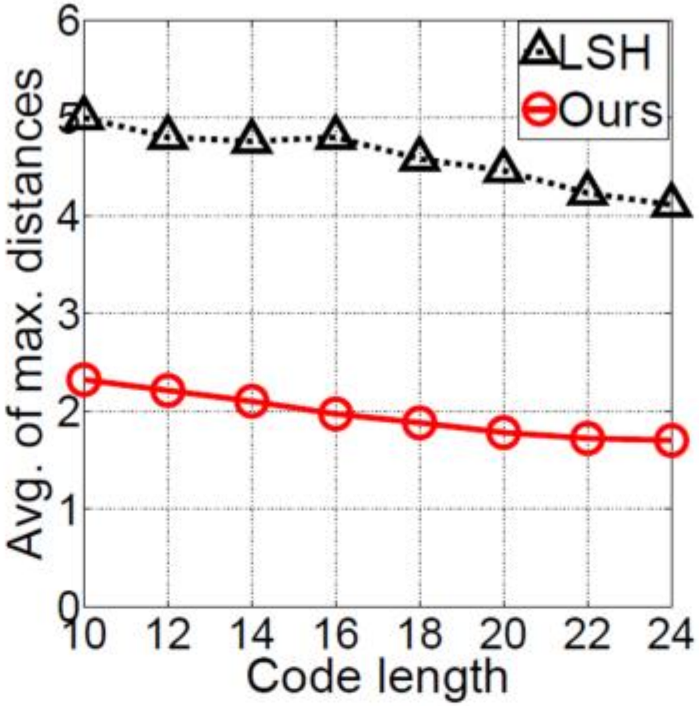




From http://sglab.kaist.ac.kr/Spherical_Hashing/

Bounding power of Hypersphere





Hyperspheres show about two times tighter bounds over the hyperplane-based approach.

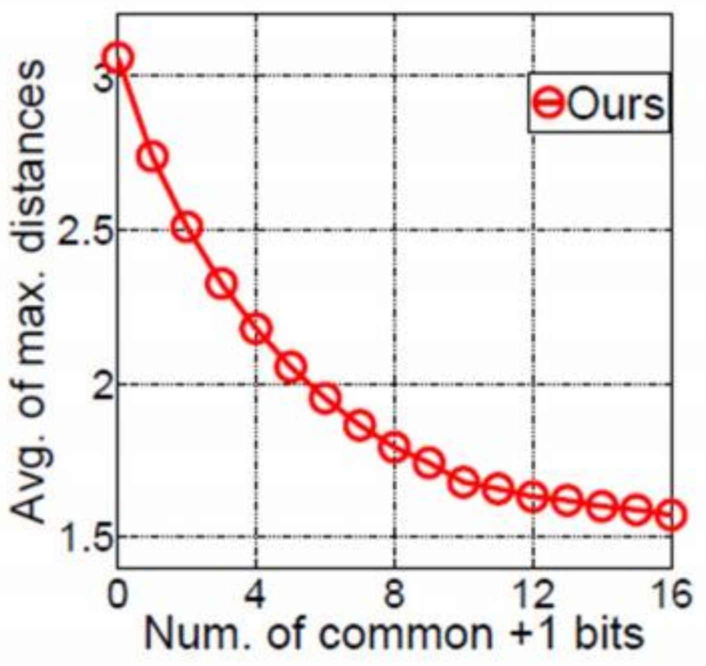
From http://sglab.kaist.ac.kr/Spherical_Hashing/

Distance between Binary Codes

Most hyperplane-based binary embedding methods use the Hamming distance $|b_i \oplus b_j|$ - does not well reflect the property related to defining closed regions with tighter bounds.

Spherical Hamming distance

$$d_{sHd} = \frac{|b_i \oplus b_j|}{|b_i \wedge b_j|}$$



Having the common +1 bits in two binary codes gives us tighter bound information than having the common 0 bits in our spherical hashing functions.

From http://sglab.kaist.ac.kr/Spherical_Hashing/

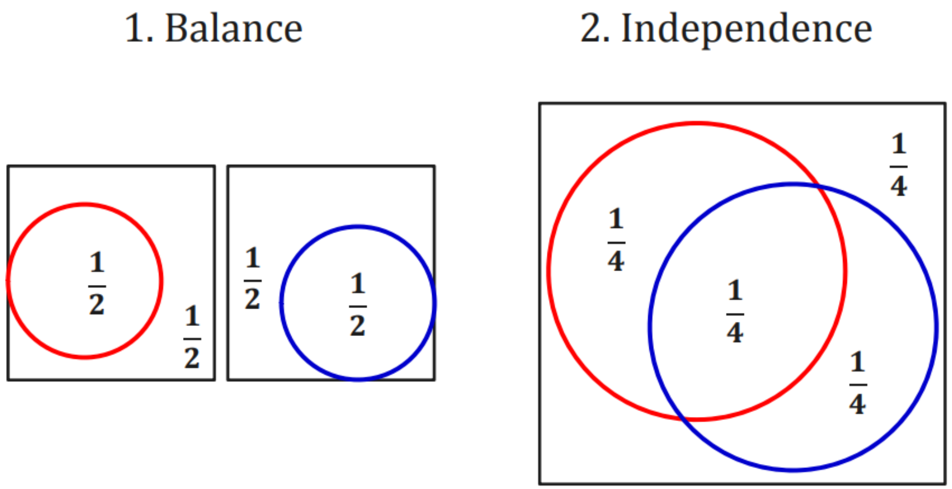
Independence between Hashing Functions

Independent hashing functions distribute points in a balanced manner to different binary codes.

Achieving such properties lead to minimizing the search time and improving the accuracy even for longer bit lengths.

Balance: $\Pr[h_k(x) = +1] = \frac{1}{2}, x \in X, 1 \le k \le c.$

Independence: $\Pr[h_i(x) = +1, h_j(x) = +1] = \Pr[h_i(x) = +1] \cdot \Pr[h_j(x) = +1] = \frac{1}{4}, x \in X, 1 \le i, j \le c.$



Iterative Optimization

Algorithm 1 Our iterative optimization process

Input: sample points $S = \{s_1, ..., s_m\}$, error tolerances ϵ_m, ϵ_s, and the number of hash functions c

Output: pivot positions $p_1, ..., p_c$ and distance thresholds $t_1, ..., t_c$ for c hyperespheres

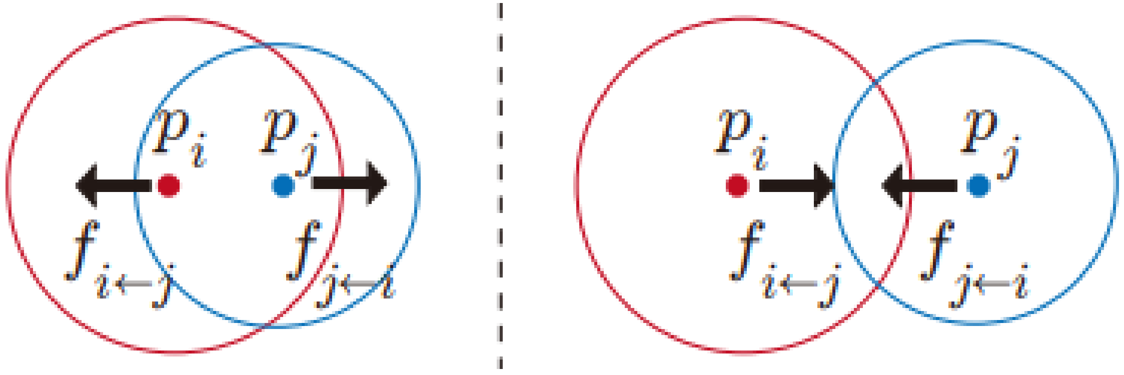
Initialize $p_1, ..., p_c$ with randomly chosen c data points from the set S
Determine $t_1, ..., t_c$ to satisfy $o_i = \frac{m}{2}$
Compute $o_{i,j}$ for each pair of hashing functions
repeat
  for $i = 1$ to $c - 1$ **do**
    for $j = i + 1$ to c **do**
      $f_{i \leftarrow j} = \frac{1}{2} \frac{o_{i,j} - m/4}{m/4} (p_i - p_j)$
      $f_{j \leftarrow i} = -f_{i \leftarrow j}$
    end for
  end for
  for $i = 1$ to c **do**
    $f_i = \frac{1}{c} \sum_{j=1}^{c} f_{i \leftarrow j}$
    $p_i = p_i + f_i$
  end for
  Determine $t_1, ..., t_c$ to satisfy $o_i = \frac{m}{2}$
  Compute $o_{i,j}$ for each pair of hashing functions
until $avg(|\ o_{i,j} - \frac{m}{4}\ |) \leq \epsilon_m \frac{m}{4}$ and $std\text{-}dev(o_{i,j}) \leq \epsilon_s \frac{m}{4}$

$$o_i = |\{s_k | h_i(s_k) = +1, 1 \leq k \leq m\}|$$

$$o_{i,j} = \left|\{s_k | h_i(s_k) = +1, h_j(s_k) = +1, 1 \leq k \leq m\}\right|, 1 \leq i,j \leq c$$

Force computation: $f_{i \leftarrow j} = \frac{1}{2} \frac{o_{i,j} - \frac{m}{4}}{\frac{m}{4}} \left(p_i - p_j\right)$ - force from p_j to p_i.

An accumulated force: $f_i = \frac{1}{c} \sum_{j=1}^{c} f_{i \leftarrow j}$.



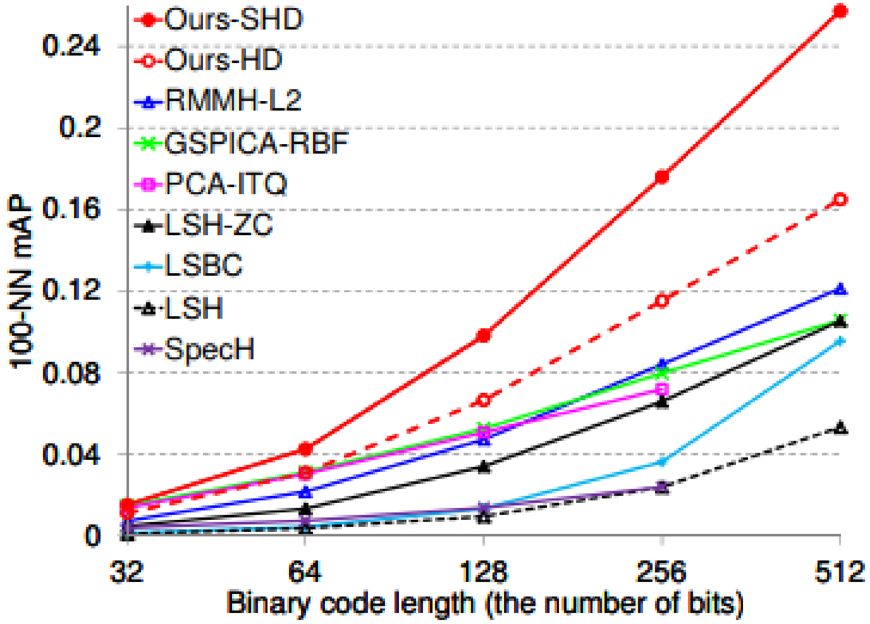
Time complexity: $O\left((c^2 + cD)m\right)$.

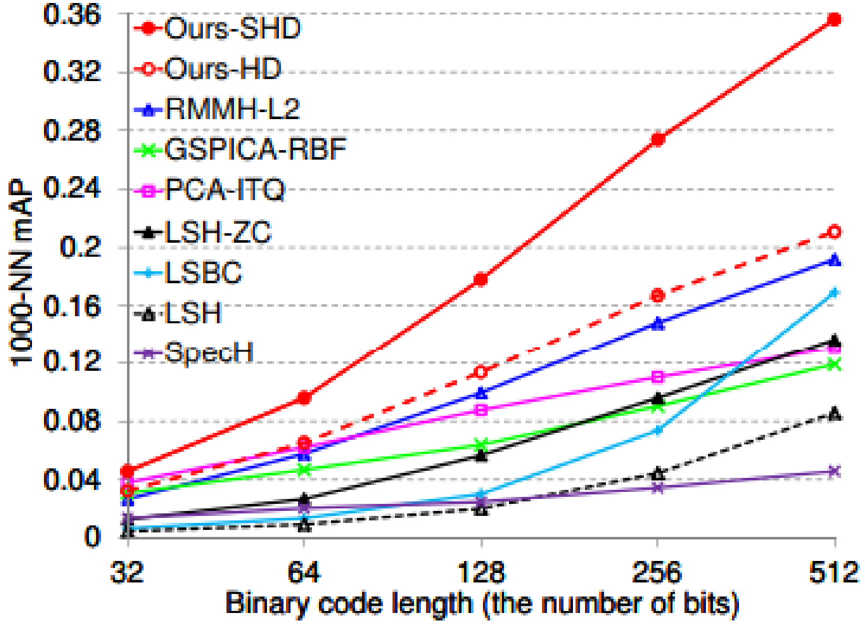
From http://sglab.kaist.ac.kr/Spherical_Hashing/

Evaluation

Three datasets:

- GIST-1M-384D
- GIST-1M-960D
- GIST-75M-384D





Randomly choose 100K data points from the original dataset as a training set for data-dependent methods.

Test: randomly chosen 1000 queries.

The performance is measured by mean Average Precision.

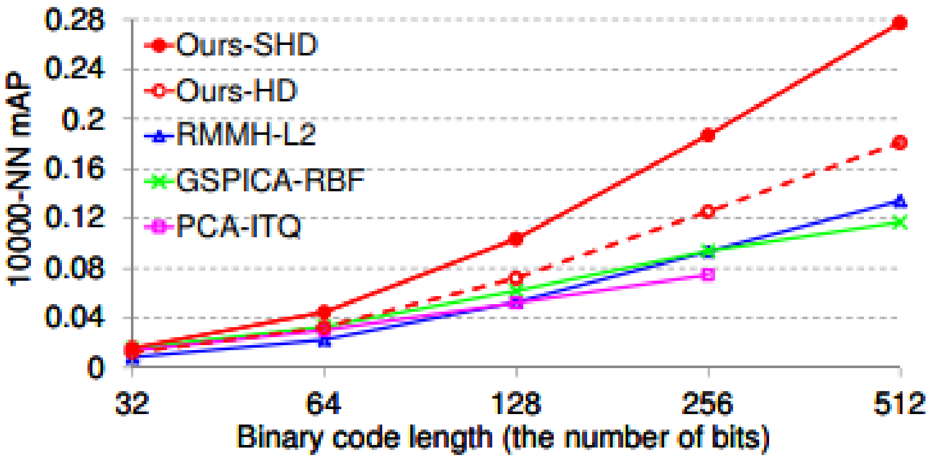
The ground truth is defined by k nearest neighbors that are computed by the exhaustive, linear scan based on the Euclidean distance.

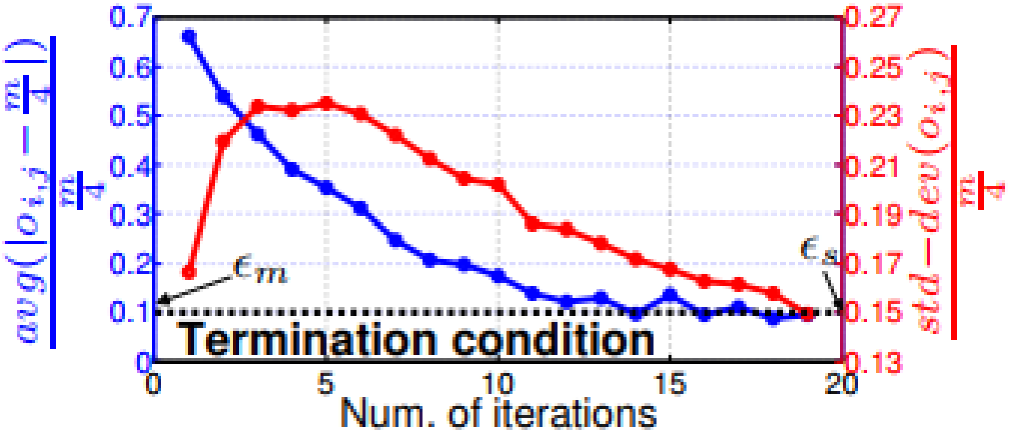
From http://sglab.kaist.ac.kr/Spherical_Hashing/

Evaluation II

Method shows significantly higher results than all the other tested methods across all the tested bit lengths even with this large-scale dataset.

Method takes 0.08 ms for generating a 256 bit-long binary code.





Conclusion

- Novel hypersphere-based binary embedding technique for providing compact data representation and highly scalable nearest neighbor search with high accuracy.
- Method significantly outperformed the tested six state-of-the-art hashing techniques based on hyperplanes with one and 75 million GIST descriptors that have 384 or 960 dimensions.

References I

- Jae-Pil Heo, Youngwoon Lee, Junfeng He, Shih-Fu Chang (2012)

Spherical hashing

In *CVPR*, pages 2957–2964

- Piotr Indyk and Rajeev Motwani (1998)

Approximate nearest neighbors: toward removing the curse of dimensionality.

In *STOC*, pages 604–613

- O. Chum, J. Philbin, and A. Zisserman (2008)

Near duplicate image detection: min-hash and tf-idf weighting.

In *BMVC*

References II

- M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni (2004)

Locality-sensitive hashing scheme based on p-stable distributions.

In *SoCG*, pages 253–262

- Alexis Joly, Olivier Buisson (2011)

Random maximum margin hashing.

In *CVPR*, pages 873-880

- Maxim Raginsky, Svetlana Lazebnik (2009)

Locality-sensitive binary codes from shift-invariant kernels

In *NIPS*, pages 1509-1517