Acta Technica *62* (2017), No. 6B, 899–912

Large data adaptive hash algorithm based on local sensitive learning

Ying Pan[1]

Abstract. In order to improve the ability of constructing tightly closed region embedded in binary codes of hash function, this paper presents a hypercircle hash binary code embedding technology based on the local edge sensitive threshold. First of all, based on hyperplane hash binary code embedding technology, this paper takes use of hypercircle to improve the hash binary code calculation process and uses multiple hypercircle to construct enclosed regions with higher dimensions; secondly, this paper designs special binary code distance measure function, namely spherical Hamming distance to calculate local edge sensitive distance threshold of hash function according to hypercircle embedding technology; finally, to achieve balanced partition and independence preserving of hash function, an iterative optimization algorithm based on pivot gravity and repulsion is designed. Compared with the classical hash binary code embedding technique, the advantages of the proposed algorithm are more obvious in high dimensional data, and more simple and easy to implement.

Key words. Hash function, Binary code, Maximum edge, Distance threshold, Hamming distance.

1. Introduction

With the development of imaging technology and related image processing technology, various kinds of new images can be generated according to different needs. These massive image databases pose challenges to the application of computer vision, especially to expandability of those applications requiring efficient similarity search. In the similarity search of low dimensional data, the nearest neighbor search and tree search algorithm are the most commonly used, but these techniques cannot be used for high dimensional data. Therefore, in order to realize the design of solution scheme of high dimensional data, binary code embedding technology has been widely studied [1, 2].

Because of the compact data representation and efficient indexing mechanism, the

[1]School of information Eigneering, Harbin University, Harbin City HeiLongJiang Province, 110051, China

high dimensional data points can be encoded into binary data using hash function to achieve higher scalability [3]. Existing hashing techniques can be broadly classified into two categories: data independence and data correlation. In the data independent technology, hash function independently selects data points [4]. Locality-Sensitive Hashing [5, 6] (Locality-Sensitive Hashing, LSH) is one of the most well known techniques in this kind of algorithm, which can be extended to a variety of hash functions. In recent years, the focus of research has shifted to considering the distribution of data points and the designing of better hash function. The most typical examples are: spectral hash function [7], semi supervised hash function [8], iterative quantization [9], joint optimization [10], and stochastic maximum marginal hash function. In all of these hash technologies, the data can be divided into two groups by hyperplane and according to the set of each data point, the data is assigned to different binary code (e.g. -1 or $+1$).

In this paper, on the basis of hyperplane, the author proposes a new scheme based on hypercircle to compute the hash binary code. Intuitively, the use of hypercircle can provide building capacity of a more powerful tightly closed region to hyperplane [11, 12]. For example, at least d+1 hyperplanes are needed to define a closed region of a d - dimensional space, and even for any high dimensional space, only one hypercircle is needed to construct such a closed region. The main contributions of the proposed algorithm in this paper are as follows: (1) This paper proposes a kind of hypercircle hash technology, and compares it with the most advanced hyperplane hash technique to verify the effectiveness of the proposed algorithm. (2) A new distance measure function of the binary code is developed for the hypercircle hash technique, which is called the spherical Hamming distance. (3) In order to realize the balanced partition and independence maintenance of hash function, an iterative optimization algorithm based on fulcrum gravity and repulsion is designed. (4) An self-adaption setting scheme of hypercircle distance threshold is designed to improve the sensitivity of local edge.

2. Hash function based on hypercircle

Use $X = \{x_1, x_2, \cdots, x_N\}$ for D dimensional space of given N data points, among which $x_i \in R^D$ represents the data points. The binary code corresponding to each data point x_i can be defined as: $b_i \in \{-1, +1\}^l$, l is the length of the code.

2.1. Binary code embedded function

The form of binary-edge-code embedded function is: $H(x) = (h_1(x), h_2(x), \cdots, h_l(x))$, among which the points in R^D is mapped to the binary cube $\{-1, +1\}^l$. Use a hypercircle to define spherical hash function. Each spherical hash function is defined by a fulcrum $p_i \in R^D$ and distance threshold $t_i \in R^+$:

$$h_i(x) = \begin{cases} -1, & d(p_i, x) > t_i \\ +1, & d(p_i, x) \leq t_i \end{cases} \tag{1}$$

Among them, $d(\cdot,\cdot)$ is Euclidean distance between two points in R^D; different distance measures (such as L_p measure) can be used to replace the Euclidean distance. The $h_i(x)$ value of each spherical hash function can be used to indicate whether the point x is located in the interior of the circle, center of which is p_i and the radius is t_i. Figure 1a gives an example of space partition, which uses three hypercircles in two-dimensional space for binary code assignment. Figure 1b is hyperplane binary coding mode.

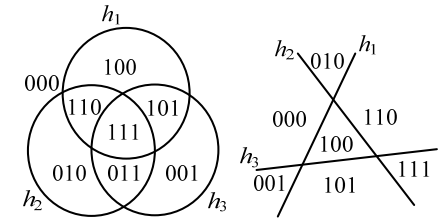

Fig. 1. Binary code embedding method

The key difference between the use of hyperplanes and hypercircles for binary codes calculation is that hyperplanes can define closed region indexed by binary code in R^D. In order to define closed region in D dimensional space, at least d + 1 hyperplanes are needed, and only one hypercircle can form closed region in any high dimensional space. In addition, compared with the use of multiple hyperplanes, it is possible to construct higher dimensional closed region by using multiple hypercircles, while the distance between points in each region is bounded. For example, by using l hypercircle, the number of bounded regions can be increased to:

$$\left(\begin{array}{c} l-1 \\ d \end{array} \right) + \sum_{i=0}^{d} \left(\begin{array}{c} l \\ i \end{array} \right). \tag{2}$$

In addition, a large sphere with a larger radius and a further center can be used to approximate the hyperplane. In the case of nearest neighbor location from the query point, it is very important to use the exact boundary to form a closed region in the nearest neighbor search. When constructing strictly enclosed region, the binary code index region of the query point can include a better candidate.

2.2. Binary code distance

Most hyperplane-based binary code embedding method uses Hamming distance [13, 14] between binary codes as bit number measure, namely $|b_i \oplus b_j|$, among which $\oplus$ is bit-exclusive-or operation, $|\cdot|$ stands for digit number of 1 in given binary code. This distance metric is used to characterize the number of hyperplanes of two given points residing on their opposite sides. However, Hamming distance is not a good way to reflect the properties of the strictly bounded closed region, which is the key problem to be solved by using the proposed hash function.

In this regard, a new distance measure for binary codes is proposed, which is

called SHD (Spherical Hamming Distance), recorded as $(d_{SHD}(b_i, b_j))$

$$d_{SHD}(b_i, b_j) = \frac{|b_i \oplus b_j|}{|b_i \wedge b_j|}. \tag{3}$$

Among them, $|b_i \wedge b_j|$ is digit of +1 between two binary codes, which can be simply calculated by using bit operation. In the proposed spherical hashing distance, two binary codes with the same +1 bits have tighter boundary information as compared with those with the common -1 bits. This is mainly because each common +1 bit represents the two data points is in interior of its corresponding hypercircle, which is as shown in figure 2.

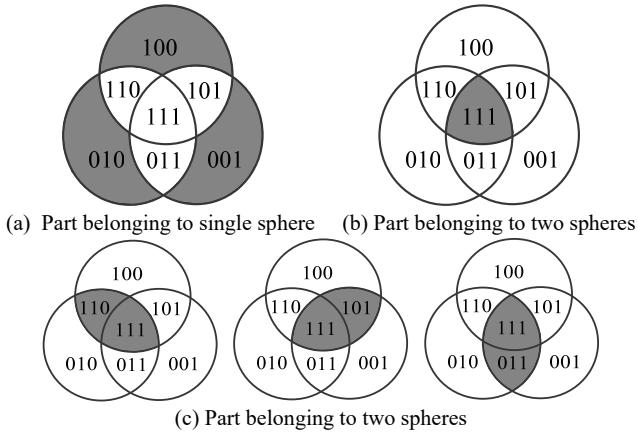

Fig. 2. Spherical Hamming distance

In order to obtain the relationship between the distance boundary and the total number of common +1 bits, the number function of the common +1 bits is used as the mean of the average distance of the data points. Therefore, to use $|b_i \wedge b_j|$ to represent is reasonable.

In the implementation process, we can attach a small value (e.g., 0.1) to avoid the denominator divided by zero. At the same time, when calculating SHD, in order to avoid the high complexity of the calculation of the division operation, the author proposes to construct a pre calculated SHD table $T(|b_i \wedge b_j|, |b_i \oplus b_j|)$, the size of which is $(l+1)^2$. Based on the distance constraints mentioned above, closed region can be defined by using common + 1 bit between two binary codes. In this closed region, we can further distinguish the distance between two binary codes based on Hamming distance $|b_i \oplus b_j|$.

2.3. Hash function independence

It is very important to realize the balanced partition of data points for each hash function, and to maintain the independence of the hash function, because independence of hash function can achieve equilibrium partitioning of data points

of different binary codes. If the above characteristics can be met, even if the length of the code is very long, high precision and search efficiency can be obtained. The form to define that each hash function h_i has the same probability for the +1 and -1 bits:

$$\Pr\left[h_i(x) = +1\right] = \frac{1}{2}, x \in X, 1 \leq i \leq l\,. \tag{4}$$

A probabilistic event V_i is defined here to represent the situation $h_i(x) = +1$, and when the conditions $\Pr\left[V_i \cap V_j\right] = \Pr\left[V_i\right] \cdot \Pr\left[V_j\right]$ are met, the event V_i and V_j meet independent relationship. Once we achieve a balanced partition for each data point (meeting the formula 4), in the case of $x \in X$ and $1 \leq i < j \leq l$, the two data bits meet the following formula:

$$\begin{aligned} &\Pr[h_i(x) = +1, h_j(x) = +1] \\ &= \Pr\left[h_i(x) = +1\right] \cdot \Pr\left[h_j(x) = +1\right] = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}\,. \end{aligned} \tag{5}$$

In general, the independence of the pairwise hash function does not guarantee the independence of that of more than three higher orders. More than two independent hash functions can also be formulated, which meets the constraint as shown in formula 4 and formula 5. But this high order independence does not improve the search quality.

2.4. Iterative optimization process

In this paper, an iterative optimization algorithm is proposed to realize the optimal calculation of l different hypercircle pivots p_i and distance thresholds t_i. The algorithm is divided into two phases:

Phase 1: Sample the data points set X to obtain a subset $S = \{s_1, s_2, \cdots, s_n\}$ of samples to achieve the approximation of the data points set X, and then randomly select l data points in the data subset S to achieve initialization of l hypercircle pivot. However, it is observed that the initial convergence rate is faster when the initial pivot is close to the training center. The main reason is that the closer the initial hypercircle is, the more overlap. In order to accelerate the algorithm, the pivot position of the hypercircle is located in the middle of the randomly selected samples, that is:

$$p_i = \frac{1}{g} \sum_{j=1}^{g} q_j\,. \tag{6}$$

Among them, q_j is randomly selected point from the data point subset S, g is the number of data points. If the value of g is too small, it is not conducive to the deployment of the pivot in the middle of the data points, and if the value of g is too large, a lot of data points will be deployed in approximate position, and computational complexity of the algorithm will be increased. Taking into account the trade-off of space, it is found through experiments that if $g = 10$, a relatively reasonable calculation results cen be obtained.

Phase 2: Refine the hypercircle pivots, and re calculate the distance threshold.

For $1 \leq ij \leq l$, set two auxiliary variables o_i and $o_{i,j}$ as:

$$o_i = |\{s_g | h_i(s_g) = +1, l \leq g \leq n\}| \,, \tag{7}$$

$$o_{i,j} = |\{s_g | h_i(s_g) = +1, h_j(s_g) = +1, 1 \leq g \leq n\}| \,. \tag{8}$$

Among them, $|\cdot|$ is the cardinality of given set. o_i is the number of $+1$ bit points in the subset S of the i^{th} hash function, which will be used to satisfy the partition balance of each ratio feature. Similarly, $o_{i,j}$ is the number of data points in the data subset S, which contains the interior points of hypercircle corresponding to the i^{th} and the j^{th} hash function. The role of $o_{i,j}$ in the iterative optimization algorithm is to guarantee the independence of the i^{th} and the j^{th} hash function. After the above two variables are calculated by data points in S subset, the optimization target is obtained by the alternating optimization of the two phases:

$$o_i = \frac{n}{2}, o_{i,j} = \frac{n}{4} \,. \tag{9}$$

Whether $o_{i,j}$ is close to or equal to $n/4$ is taken as the standard here to achieve the adjustment of the two hypercircle pivots. Intuitively, for i and j of each hypercircle, when $o_{i,j}$ is larger than $n/4$, set force of repulsion between the two hypercircle pivots to make them far away from each other; otherwise exert a force of attraction to make them close to each other. Secondly, once the pivot is calculated, the distance threshold t_i of the i^{th} hypercircle is adjusted to make o_i close to $n/2$ to meet equilibrium partitioning of hypercircle data points.

Under the premise of satisfying the above constraints, the iterative optimization process is carried out until the obtained hypercircle is no longer improved. The ideal mean and standard deviation of $o_{i,j}$ is distributed as $o_{i,j}$ and $n/4$. However, in order to avoid overfitting, the mean value and standard deviation threshold value of the optimization algorithm are set up respectively as $\varepsilon_m\%$ and $\varepsilon_s\%$, and according to the experimental results, $\varepsilon_m\% = 10\%$ and $\varepsilon_s\% = 15\%$ are selected here.

Force calculation: definition of the force $f_{i \to j}$ (attraction or repulsion) of p_i on p_j is as shown in figure 3, the specific form is:

$$f_{i \leftarrow j} = \frac{1}{2} \frac{o_{i,j} - n/4}{n/4} (p_i - p_j) \,. \tag{10}$$

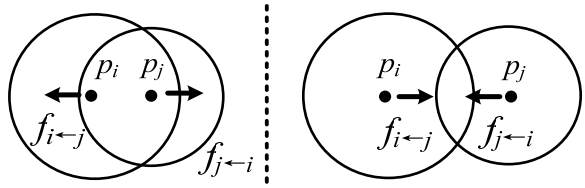

Fig. 3. Force calculation sketch map

The acceleration force f_i can be expressed as the mean value calculated of the other pivot forces in the form of:

$$f_i = \frac{1}{l} \sum_{j=1}^{l} f_{i \leftarrow j} \,. \tag{11}$$

If the acceleration force f_i is applied in p_i, p_i can be updated to $p_i + f_i$. The proposed iterative optimization process is shown in algorithm 1.

Algorithm 1 iterative optimization process

Input: Sample points $S = \{s_1, s_2, \cdots, s_n\}$, error tolerances are ε_m and ε_s, and the number of hash functions is l;

Output: Pivot position $p_1, p_2, \cdots, p_l$, distance threshold $t_1, t_2, \cdots, t_l$ of hypercircle l;

Use l randomly selected data points in the S to initialize the pivot position $p_1, p_2, \cdots, p_l$;
$p_i = \frac{1}{g} \sum_{j=1}^{g} q_j$
Based on section 2.5,determine $t_1, t_2, \cdots, t_l$, and meet $o_i = n/2$;
Calculate $o_{i,j}$ for each pair of hash functions;
repeat
  for i = 1: l - 1 **do**
    for j = i + 1: l **do**
      $f_{i \leftarrow j} = \frac{1}{2} \frac{o_{i,j} - n/4}{n/4} (p_i - p_j)$;
      $f_{j \leftarrow i} = -f_{i \leftarrow j}$;
    end for
  end for
  for i = 1: l **do**
    $f_i = \frac{1}{l} \sum_{l}^{1} f_{i \leftarrow j}$;
    $p_i = p_i + f_i$;
  end for
  Based on section 2.5,determine $t_1, t_2, \cdots, t_l$and meet $o_i = n/2$;
  Calculate $o_{i,j}$ for each pair of hash functions;
until $avg(|o_{i,j} - \frac{n}{4}|) \leq \varepsilon_m \frac{n}{4}$, $std - dev\,(o_{i,j}) \leq \varepsilon_s \frac{n}{4}$

The computational complexity of the iterative optimization process above is $O\left(\left(l^2 + lD\right) n\right)$. In fact, the iterative process of the proposed algorithm can be completed within 30 times. In addition, its overall computing time is less than 30 seconds even for a long code of 128 bits.

2.5. Local edge sensitive distance threshold calculation

In each iteration computation step, it is necessary to determine the sensitivity threshold $t_1, t_2, \cdots, t_l$ of the local edge in order to meet $o_i = n/2$ to achieve the equalization of the partition. In this regard, distance $d(p_i, s_n/2)$ between each t_i and $s_n/2$ can be simply set, and the samples in S can be sorted as $s_1, s_2, \cdots, s_n$ according to the distance of p_i. However, this simple approach may lead to an unreasonable partition, especially when $s_n/2$ is located in a dense region.

In this regard, this paper proposes a threshold optimization method based on the maximum edge to firstly sort the distance $d\left(p_i, s_j^s\right)$ between samples in S and pivots as $s_1^s, s_2^s, \cdots, s_n^s$. The improved form proposed in this paper is to obtain set J by optimizing using the candidate points at the middle point $n/2$:

$$J = \left\{ j | (\frac{1}{2} - \beta)n \leq j \leq (\frac{1}{2} + \beta)n, j \in Z^+ \right\}. \tag{12}$$

Among them, β is the tolerance parameter to control the standard of equilibrium partition. Set $\beta = 0.05$ here, and then calculate the sort list $\hat{j}$ of sample indexes that can maximize the edge of hypercircle:

$$\hat{j} = \arg\max d(t_i, s_{j+1}^s) - d(t_i, s_j^s). \tag{13}$$

The value of the distance threshold t_i can be calculated according to hyperplane partition $s_{\hat{j}}^s$ and $s_{\hat{j}+1}^s$, which can be calculated as follows:

$$t_i = \frac{1}{2}(d(t_i, s_{\hat{j}}^s) + d(t_i, s_{\hat{j}+1}^s)). \tag{14}$$

3. Performance evaluation

In this section, the author will analyze the performance of the proposed algorithm, the experimental hardware is set as: Xeon X5690 machine with the memory size of 24GB, which can store complete data set.

3.1. Test data set

During the experiment, the following four data sets were selected as the test object, specific information is as follows:

(1) GIST-1M-384D data set, 384 dimensional set, including one million GIST descriptors of small image subsets.

(2) GIST-1M-960D data set, 960 dimensional set, also including one million GIST descriptors.

(3) GIST-75M-384D data set, 384 dimensional set, including 75 million GIST descriptors of 80 million small image subsets.

(4) ILSVRC data set, one million 1000 dimensional descriptors, which are a subset of the ImageNet database.

(5) VLAD-1M-8192D data set, one million 8192 dimensional VLAD descriptors (128 dimensional SIFT features and 64 codebook vector).

3.2. Performance testing

Randomly select a subset of 1000 queries in GIST-1M-384D, GIST-1M-960D and VLAD-1M-8192D databases, randomly select a subset of 500 queries in GIST-75M-384D, data points of these subsets do not overlap each other. The mean accuracy index (mAP) was used to evaluate the algorithm. When calculating the accuracy index, it is considered that all items have low or equal Hamming distance from the given query. The contrast algorithm is selected as follows:

(1) LSH and LSH-ZC algorithm, respectively represent the local sensitive hash algorithm with zero center data points. (2) LSBC algorithm, local sensitive binary code, the bandwidth parameter used in the experiment is the reciprocal of the average distance between the data points in the data set. (3) SpecH algorithm, spectral hashing algorithm. (4) PCA-ITQ algorithm, iterative quantization algorithm. (5) RMMH-L2 algorithm, random maximum margin hashing (RMMH) algorithm with triangular L2 kernel, the parameter M of number of samples of each hash function is set as 32. (6) GSPICA-RBF algorithm, generalized similarity preserving independent component analysis (GSPICA) algorithm with RBF kernel function, in which the parameters γ and P are set as 1. (7) Ours-HD and Ours-SHD algorithms, respectively representing taking the algorithm steps in this paper and using the common Hamming distance (Ours-HD) and the binary code distance (Ours-SHD) in this paper. Ours-HD and Ours-SHD algorithms use threshold segmentation method based on the maximum edge distance.

For the above hash method, 100k parameters are randomly selected from the original data set as the training set to estimate algorithm parameters. For each algorithm, 5 experiments were performed to obtain more statistically significant results. Figure 4 shows the mAP index of the nearest neighbor query results on the GIST-1M-384D dataset when $k = 100$. Figure 5 shows the mAP index of the nearest neighbor query results on the GIST-1M-960D dataset when $k = 100$. Figure 6 shows the mAP index of the nearest neighbor query results on the ILSVRC dataset $k = 100$. Figure 7 shows the mAP index of the nearest neighbor query results on the VLAD-1M-8192D dataset $k = 100$.

Figure 4 shows that the mAP index of Ours-SHD algorithm is better from the 32 bit to the 512 bit, and the advantages of this algorithm are more obvious with the increase of bit length. The main reason is that the multiple hypercircles used in this paper can be more effective than the plane closed region to create tight distance boundary. In Figure 8, for the given 0.1mAP index, this paper uses 128 bits to encode each image, while the other methods are more than 256 bits. Therefore, this algorithm is 2 times more compact than other algorithms. Once the nearest neighbor image is determined based on the binary code, the binary code can be used for additional reordering of these images. The results of Figure 5$\sim$ Figure 7 show similar results with Figure 4, because the ILSVRC dataset and the VLAD-1M-8192D dataset are large, in the experiments in Figure 6$\sim$7, only part of the algorithm is

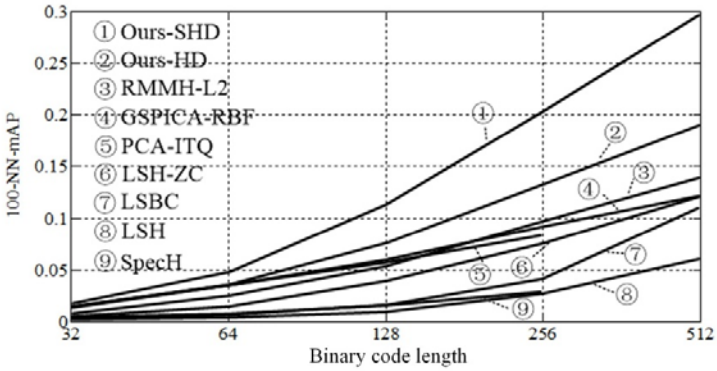

Fig. 4. Comparison of mAP index (GIST-1M-384D)

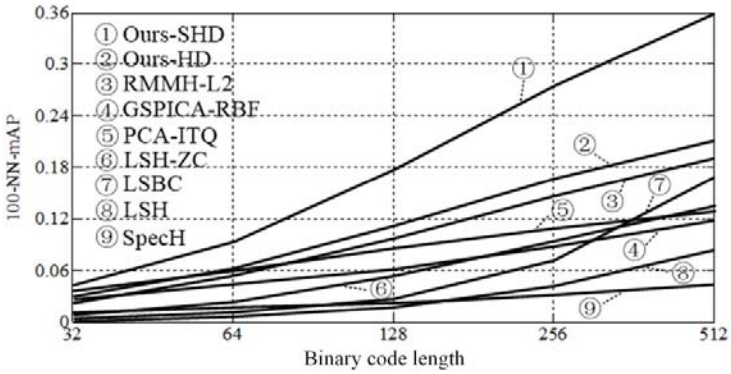

Fig. 5. Comparison of mAP index (GIST-1M-960D)

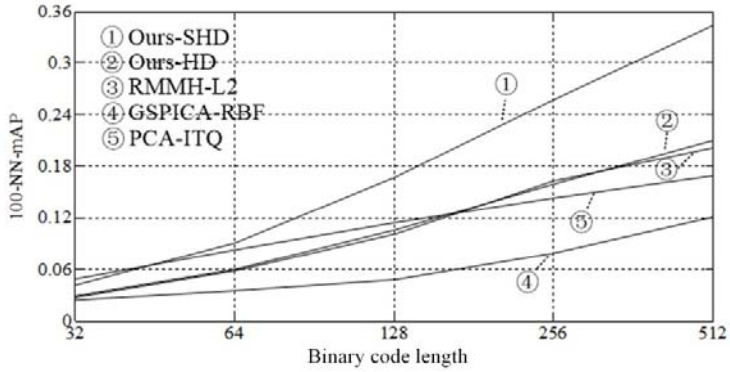

Fig. 6. Comparison of mAP index (ILSVRC)

selected for the experiment.

When assigning 64 bit coding for each image, Figure 8 shows recall curve of different methods for different number of hash tables in GIST-75M-384D data set,

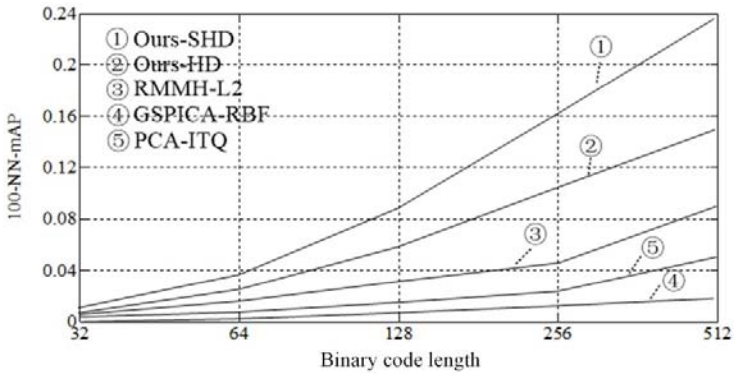

Fig. 7. Comparison of mAP index (VLAD-1M-8192D)

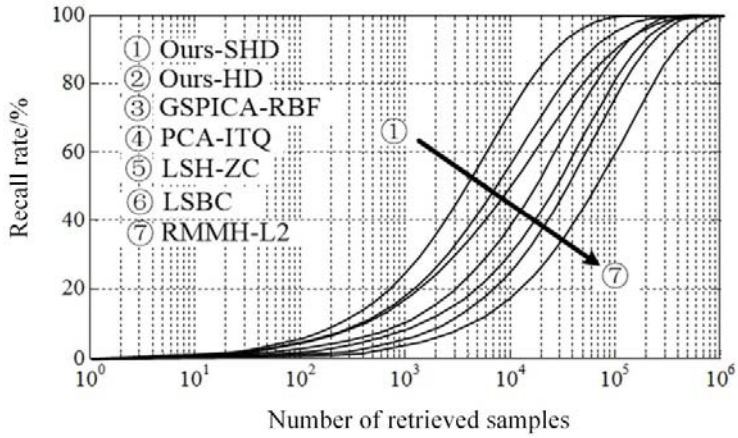

Fig. 8. Recall rate index comparison (GIST-75M-384D)

Ours-SHD, RMMH-L2, GSPICA-RBF, PCA-ITQ, LSH-ZC, LSBC and Ours-HD algorithms are selected as comparison algorithms. Table 1 gives a comparison of the computational time of the proposed algorithm and the selected algorithms in the 32 bit to the 512 bit.

According to the results of Figure 8, we can see that the performance of the proposed algorithm is better than that of other algorithms because it uses more hash tables, which reflects the good checking performance of the algorithm.

The experimental results of Table 1 show that in calculation efficiency the calculation time of the proposed algorithm distributes between 3.2∼9.3s, the calculation time of Ours-HD algorithm distributes between 3.1∼9.8s, the computation time of the two algorithms is superior to several other comparison algorithms, the computing time is far longer than that of the proposed algorithm in this paper, based on which the advantage of the proposed algorithm in computational efficiency is verified.

Table 1. Comparison of computation time on GIST-1M-384D data set (*10s)

algorithm	32	64	128	256	512
Ours-SHD	33.2	**44.5**	**5.8**	**7.6**	**9 .3**
LSBC	88.6	110.3	13.4	18.5	26.8
SpecH	77.8	99.6	12.8	17.3	23.4
PCA-ITQ	99.3	112.3	18.6	28.3	39.4
RMMH-L2	88.1	99.6	12.6	19.3	24.8
GSPICA-RBF	66.8	88.9	11.5	15.8	19.4
Ours-HD	**33.1**	44.8	66.4	8.6	9.8

4. Conclusion

This paper proposes a hypercircle hash binary code embedding technology based on local edge sensitive threshold, uses hypercircle to improve hash binary code calculation process, uses multiple hypercircles to construct closed region of high dimension, designs special binary code distance measure function and iterative optimization algorithm based on pivot attraction and repulsion, the experimental results verify the effectiveness of the proposed algorithm. In the next step, the research direction is to expand the arbitrary kernel function of the hypercircle hash binary code embedding technique.

Acknowledgement

Design and implementation of ARP spoofing defense system of campus network (12533041).

References

[1] Zhang X, Cui Y, Li D, et al.: (2012) *An adaptive mean shift clustering algorithm based on locality-sensitive hashing*[J]. Optik - International Journal for Light and Electron Optics, 123(20):1891–1894.

[2] Ali S H A, Fukase K, Ozawa S: (2016) *A fast online learning algorithm of radial basis function network with locality sensitive hashing*[J]. Evolving Systems, 7(3):1-14.

[3] Cao Y, Liu F, Cai X: (2013) *Research on high dimension image data indexing technology based on locality sensitive Hashing algorithm*[J]. Journal of Liaoning University of Technology, 33(1).

[4] Nehme R V, Works K, Lei C, et al.: (2013) *Multi-route query processing and optimization*[J]. Journal of Computer & System Sciences, 79(3):312-329.

[5] Wang D, Chai K Y: (2011) *Exploring Locality of Reference in P2P VoD Systems*[C]// Global Telecommunications Conference. IEEE Xplore, 2011:1-6.

[6] Zhou Y, Liu C, Li N, et al.: (2016) *A novel locality-sensitive hashing algorithm for similarity searches on large-scale hyperspectral data*[J]. Remote Sensing Letters, 7(10):965-974.

[7] Carli L D, Sommer R, Jha S: (2014) *Beyond Pattern Matching:A Concurrency Model for Stateful Deep Packet Inspection*[J]. 28(4):1378-1390.

[8] Nissim N, Cohen A, Glezer C, et al.: (2015) *Detection of malicious PDF files and directions for enhancements: A state-of-the art survey*[J]. Computers & Security, 48(C):246-266.

[9] Abhirama M, Bhaumik S, Dey A, et al.: (1991) *Stability-conscious Query Optimization*[J]. Electrochimica Acta, 36(10):1579-1583.

[10] Wang X, Wang X, Varma R K, et al.: (2009) *Selection of hyperfunctional siRNAs with improved potency and specificity.*[J]. Nucleic Acids Research, 37(22):e152.

[11] Duan S, Thummala V, Babu S: (2009) *Tuning database configuration parameters with iTuned*[J]. Proceedings of the Vldb Endowment, 2(1):1246-1257.

[12] Li F, Klette R: (2008) *Adaptive Mean Shift-Based Clustering*[J]. Lecture Notes in Computer Science, 40(11):1002-1009.

[13] Duan S, Fokoue A, Hassanzadeh O, et al.: (2012) Instance-Based Matching of Large Ontologies Using Locality-Sensitive Hashing[J]. 7649:49-64.

[14] He Z, Wang Q: (2008) *A Fast and Effective Dichotomy Based Hash Algorithm for Image Matching*[C]// Advances in Visual Computing, International Symposium, Isvc 2008, Las Vegas, Nv, Usa, December 1-3, 2008. Proceedings. DBLP, 2008:328-337.

[15] Zhu Z, Xiao J, He S, et al.: (2015) *A multi-objective memetic algorithm based on locality-sensitive hashing for one-to-many-to-one dynamic pickup-and-delivery problem*[J]. Information Sciences, 329:73-89.

Received May 7, 2017