Double-Bit Quantization and Index Hashing for Nearest Neighbor Search

Hongtao Xie, Zhendong Mao, Yongdong Zhang, *Senior Member, IEEE*, Han Deng, Chenggang Yan, and Zhineng Chen

***Abstract*—As binary code is storage efficient and fast to compute, it has become a trend to compact real-valued data to binary codes for the nearest neighbors (NN) search in a large-scale database. However, the use of binary code for the NN search leads to low retrieval accuracy. To increase the discriminability of the binary codes of existing hash functions, in this paper, we propose a framework of double-bit quantization and index hashing for an effective NN search. The main contributions of our framework are: first, a novel double-bit quantization (DBQ) is designed to assign more bits to each dimension for higher retrieval accuracy; second, a double-bit index hashing (DBIH) is presented to efficiently index binary codes generated by DBQ; and third, a weighted distance measurement for DBQ binary codes is put forward to re-rank the search results from DBIH. The empirical results on three benchmark databases demonstrate the superiority of our framework over existing approaches in terms of both retrieval accuracy and query efficiency. Specifically, we observe an absolute improvement on precision of 10%–25% in most cases and the query speed increases over 30 times compared to traditional binary embedding methods and linear scan, respectively.**

***Index Terms*—Nearest neighbor search, double-bit quantization, double-bit index hashing, weighted distance measurement, binary embedding.**

I. Introduction

NEAREST neighbor (NN) search has been one of the key problems of visual applications including image retrieval [1]–[4], object recognition [5]–[8] and copy detection [9]–[12]. Further, NN search is irrelevant to feature extraction method exploited, while the latest deep learning based methods [13]–[15] gradually replace the traditional feature extraction methods, such as SIFT [16], [17] and GIST [18], [19]. NN search consists in finding the closest matches of a given query signature in large amounts of reference signatures [20]–[22]. When searching similar signatures in a large-scale database composed of floating-point values, it usually computes the Euclidean distance between the query and all the reference signatures, which is quite costly. As a consequence, handling large quantities of data has become a challenge on its own.

Generally, the problem can be solved from two aspects: using binary codes to represent image signatures, which reduces the matching time between signatures and saves the memory storage; constructing index structure to organize the compact signatures to improve the retrieval efficiency. Although a lot of existing works concentrate on binary embedding and binary code indexing, there still exist some issues on both of them for NN search as follows:

1) Binary embedding leads to low retrieval accuracy. Binary codes take up less storage and require only a small amount of machine instructions when comparing for similarities. Thus, the use of binary code can greatly accelerate the nearest neighbor search and signature matching. Even with the most time-consuming linear search method, millions of binary codes can be compared to a query in less than a second [32]. Despite of its remarkable advantages, the drawbacks are not ignorable: on one hand, mapping floating-point data to binary codes results in obvious loss of information, as a single floating-point value is usually converted into one-bit binary code (0 or 1); on the other hand, it also evidently reduces the distinctiveness between different signatures [43].
2) Binary code indexing cannot handle long codes effectively. Unlike real-valued signatures, binary codes cannot be directly stored or organized by traditional index structures, such as tree-based index [23]–[26] and locality sensitive hashing (LSH) [27]–[30], to guarantee retrieval efficiency. But binary code itself can be directly used as index address to create a hash table to be indexed, which increases the search speed compared to the linear scan. However, the use of binary code as a direct hash index is not necessarily effective in many cases, as it needs to search all the buckets in hamming balls around the query in order to find the nearest neighbors in Hamming space.

Manuscript received September 6, 2017; revised January 26, 2018, June 3, 2018, and August 12, 2018; accepted September 10, 2018. Date of publication October 1, 2018; date of current version April 23, 2019. This work was supported in part by the National Key Research and Development Program of China (2017YFC0820600); in part by the National Defense Science and Technology Fund for Distinguished Young Scholars under Grant 2017-JCJQ-ZQ-022; in part by the National Nature Science Foundation of China under Grants 61525206, 61771468, 61772526, and 61502477; and in part by the Youth Innovation Promotion Association Chinese Academy of Sciences (2017209). The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Benoit Huet. (*Corresponding author: Zhendong Mao.)*

H. Xie, Z. Mao, and Y. Zhang are with the School of Information Science and Technology, University of Science and Technology of China, Hefei 230026, China (e-mail: htxie@ustc.edu.cn; maozhendong2008@gmail.com; zhyd73@ustc.edu.cn).

H. Deng is with the Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China (e-mail: denghan@iie.ac.cn).

C. Yan is with the Institute of Information and Control, Hangzhou Dianzi University, Hangzhou 310000, China (e-mail: cgyan@hdu.edu.cn).

Z. Chen is with the Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China (e-mail: zhineng.chen@ia.ac.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMM.2018.2872898

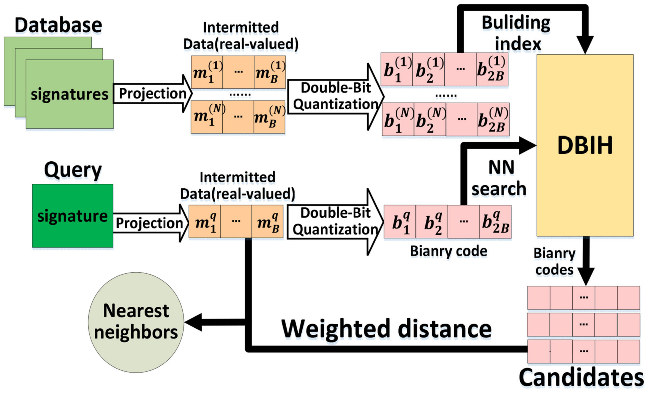


Fig. 1. Framework of the proposed double-bit quantization and index hashing method. Each signature is first projected into *B*-dimensional real-valued intermediate data. Then double-bit quantization converts intermediate data into 2*B*-bit binary codes. Next, DBIH selects the top-*k* nearest neighbors in hamming space as candidates. Finally, the weighted distance measurement is used to re-rank candidates to get final results.

Indeed, the number of hash buckets grows exponentially with the search radius. Even if the radius of the search is small, the number of buckets to be checked is usually greater than the number of signatures in the database [32]. So the query efficiency is much lower than that of linear scan. In general, when the binary code is longer than 32-bit, the query efficiency is lower than linear scan. However, the binary signatures are usually much longer than 32 bits long [32]. Thus, how to deal with long codes to build binary code index remains a key problem for NN search.

Building on the previous analysis, the framework of double-bit quantization and index hashing (DBQ-IH) is proposed in this paper, as demonstrated in Fig. 1. Firstly, all the signatures in database are mapped to binary codes by double-bit quantization (DBQ), to increase the discriminability of the binary codes of existing hash functions. Then the double-bit index hashing (DBIH) is built to organize those binary codes. For a given query, it will be mapped to DBQ binary code in the same way, and its real-valued data is preserved as well. Next, the results of top-*k* similar binary codes in hamming space are selected from DBIH. In the following, the weighted distance measurement (WDM) is applied to the results for re-ranking. Finally, the nearest neighbors are achieved through the above processes.

Summarily, the main contributions of the framework are:

- Double-bit quantization. We map each dimension of data to double-bit rather than one-bit for higher retrieval accuracy. Meanwhile, the hamming distances between binary codes based on DBQ are assigned different weights according to their spatial relationship.
- Double-bit index hashing. A new binary code indexing is brought up based on multi-index hashing (MIH) [32] for DBQ binary codes to speed up NN search, without changing the structure of the original MIH.
- Weighted distance measurement. WDM is proposed to compute the distance between reference DBQ binary codes and uncompressed query signatures to re-rank the nearest neighbors from DBIH for higher accuracy, as uncompressed query signatures have more discriminability than binary codes.

For experimental comparisons, the proposed framework is evaluated on three benchmark datasets, including the SIFT [16], [17] signatures and GIST [18], [19] signatures in BIGANN [11] and GIST signatures in Caltech101 [33]. It demonstrates that the proposed DBQ and WDM consistently and evidently improve the retrieval accuracy over the traditional binary embedding methods. Meanwhile, DBIH obviously accelerates NN search compared to linear search. In some cases, DBQ-IH can increase the precision on the order of 10%∼25% against original methods and accelerate the NN search by over 30 times compared to the exhaustive linear scan.

The rest of the paper is organized as follows. Section II gives a review of related works. In Section III, we elaborate the approaches of the framework DBQ-IH. Section IV presents the experimental results on BIGANN and Caltech101 for NN search. Finally, conclusions are given in Section V.

II. Related Work

In this section, we first introduce the binary embedding methods. Then, the binary code indexes are briefly described. Finally, the weighted distance algorithms are analyzed in detail.

A. Binary Embedding

The prominent advantages of binary codes lead to the explosion in binary embedding techniques. Binary embedding aims to transform real-valued signatures into binary codes, while it guarantees that similar signatures are mapped into the same binary codes with a high probability. There are various existing binary embedding methods. Principal component analysis (PCA) [33], [34] is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables, to reduce the dimension of correlated coefficients. The locality sensitive hashing (LSH) [31], [36], [37] measures the similarity between two vectors using the inner product operation. The hashing function of LSH projects the raw features into hyper-planes, and the coefficients of which are drawn from the multivariate normal distribution. The spectral hashing (SH) [38], [39] finds the best codes for given feature vectors by solving the optimization problem, which is expressed by the sum of weighted differences between raw feature vectors. The key idea of iterative quantization (ITQ) [40], [41] simply rotates the data to minimize the error that is defined by the difference between the binary hashing code and the low dimensional vector acquired in dimensionality reduction.

In general, the binary embedding algorithms can be decomposed into two steps: 1) The signatures are first embedded in an intermediate space and 2) Thresholding is performed in intermediate space to obtain binary outputs. In order to express the meaning of binary embedding clearly, we introduce a set of notations. Let s be an image signature with K dimensions in space Ω and let h_k be a binary embedding function, i.e., $h_k : \Omega \to \{0, 1\}$. A set $H = \{h_k, k = 1 \dots K\}$

of K functions define a multidimensional embedding function $h : \Omega \rightarrow \{0, 1\}^K$ with $h(s) = [h_1(s)\ h_K(s)]'$. Note that real-valued signatures are not directly converted into binary codes via binary embedding. For LSH, SH, PCAE and PCAE-ITQ, binary embedding function h_k can be decomposed as follows:

$$h_k(s) = q_k\left[g_k(s)\right], \tag{1}$$

where $g_k(s) : \Omega \rightarrow \mathcal{R}$ (the intermediated space) is projection function and $q_k(s) : \mathcal{R} \rightarrow \{0, 1\}$ is quantization function. That is, binary embedding firstly projects image signature s to real-valued multidimensional vector $g(s) = [g_k(s), k = 1 \ldots K]'$, which is an extremely good approximation to the original signature. Then, the real-valued data will be quantized into binary codes by thresholding (0 is often set as the threshold). Namely, if $g_i(s) > 0$, s_i is mapped to 1. Otherwise, s_i will be mapped to 0. Thus, traditional quantization function just roughly divides each dimension into two parts, decoded as 0 or 1, which considerably reduces the discriminability [43].

Up to now, many binary embedding methods [47]–[55] have been proposed. The basic idea of MH [47] is to encode each projected dimension with multiple bits of natural binary code, based on which the Manhattan distance between points in the hamming space is calculated. The method in [48] quantizes each projected dimension into double bits with adaptively learned thresholds. Neighborhood Preserving Quantization (NPQ) [49] assigns multiple bits per hyper plane based upon adaptively learned thresholds. Variable Bit Quantization (VBQ) [50] provides a data driven non-uniform bit allocation across hyper planes. Zhu *et al.* [51] propose a novel approach based on [48], which can efficiently handle the situation under non-Gaussian distribution input. Xiong *et al.* [52] provide a novel adaptive quantization (AQ) strategy that adaptively assigns varying numbers of bits to different hyper planes according to their information content. Hamming Compatible Quantization (HCQ) [53] preserves the capability of similarity metric between Euclidean space and Hamming space by utilizing the neighborhood structure of raw data. Those hashing methods use supervised learning to get higher accuracy, which bring extra time overhead.

B. Binary Code Indexing

Linear search [41] is an intuitive and typical exact search method for binary codes. Given a query, brute-force matching is performed to find the nearest neighbors. However, using linear search for matching becomes a bottleneck for large datasets [43]. Therefore, attention has been paid to the hierarchical clustering trees (HCT) [42] for higher efficiency. HCT selects cluster centers randomly and builds indexes with the entire binary codes. However, this degrades search performance when code gets longer. To deal with long codes, M. Nououzi *et al.* propose the multi-index hashing method (MIH) [32]. The main idea is to divide the long code into several shorter codes and use these segments as address to build multiple hash tables. To save more memory usage and get higher search efficiency, the inverted multi-index structure [1] is designed by taking the code distributions among different bits into account for index construction. Among them, the notion of MIH is widely used in many methods for indexing binary codes, as it divides long codes into substrings and builds multiple hash tables, which dramatically accelerates the retrieval efficiency.

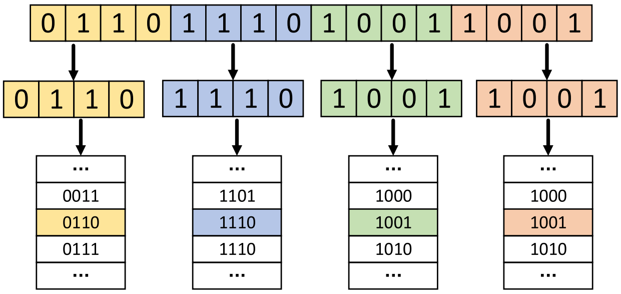


Fig. 2. Multi-index hashing.

Supposing we have a dataset of N binary codes $\{p_1, p_2, \ldots p_N\}$, and each of them is B-bit long. When a binary code is used directly as an index value, the run-time of NN search is determined by the number of hash buckets to be checked, which is $\sum_{i=0}^{R} C_B^i$ in total, where R stands for the hamming distance. Thus, the size of the Hamming space to be searched increases exponentially as the hamming distance increases. Therefore, the number of hash buckets to be examined even greater than the size of the database in many cases. In MIH, as demonstrated in Fig. 2, each B-bit binary code is divided into m disjoint binary substrings, the length of which is $b = B/m$. Then it inserts each substring as an index into m different hash tables. Given a query vector, a hash bucket close to the query in at least one such hash table is considered a neighbor candidate, that is

$$\left\| q^k - p_i^k \right\| \leq R/m, \tag{2}$$

where k means the index of the segment of the origin binary code. As a result, the number of hash buckets to be examined decreases to $\sum_{k=1}^{m} \sum_{i=0}^{R/m} C_b^i$. Then, the entire binary code is used to check the validity of the candidates to exclude signatures that are not nearest neighbors. Finally, the obtained results are the exact nearest neighbors of the query in Hamming space.

C. Weighted Distance

Since binary representation leads to the loss of information [46], weighted distance is proposed to addresses this concern. In [44], the authors attempt to weight the hamming distance of local signatures for image matching. In [45], two weights are assigned to each hash bit and a score function is defined to measure the similarity between binary signatures in developing a ranking algorithm. To compute the distance between different spaces, Albert *et al.* propose the asymmetric distance algorithm [44], as the distance is calculated between Hamming space and Euclidean space. They note that the compressing query signature is not necessary when it comes to NN search. In practice, the memory storage of a single non-binary signature is negligible. In addition, the distance between the original signature and the compressed signature can be efficiently calculated through the lookup tables. The main advantage of asymmetric algorithms is that they can achieve higher search accuracy as they use the more accurate position information of the query.

In [44], an algorithm for asymmetric distance based on the expected value is proposed, which is defined as:

$$d_E\left(x,\ y\right) = \sum_k d(g_k\left(x\right),\ E[g_k\left(u\right)|h_k\left(u\right) = h_k\left(y\right)]) \quad (3)$$

where $d(.)$ is the Euclidean distance, and E means the expectation value. d_E is simply computed as the Euclidean distance between the query and the corresponding expected value of the intermediated data $g_k(u)$, such that $h_k(u) = h_k(y)$. As the distance can be calculated through pre-computed look-up tables, the increase of time complexity is negligible.

III. The Proposed Method

In this section, we start with a general overview of DBQ-IH. Then, the novel DBQ, DBIH and WDM are precisely described.

A. Overview

The framework of DBQ-IH is a binary based system for fast large-scale nearest neighbor search. After feature extraction process, binary embedding methods are used to map the floating-point signatures to binary codes for higher retrieval efficiency and less memory storage. Instead of utilizing traditional quantization, we use double-bit quantization to convert the intermediate data in Euclidean space to binary codes for more discriminative binary signatures. Then double-bit index hashing is designed to fit DBQ binary codes. DBIH creates multiple index hash tables for each segment of binary codes. For a given query, we not only convert it to binary code, but also preserve its original signature. In this way, more information of the query can be utilized to improve the retrieval accuracy by re-ranking the results from DBIH.

Therefore, the whole framework is divided into three parts to be introduced. The first is the method of double-bit quantization, followed by the double-bit index hashing, and the weighted distance measurement is detailed at last.

B. Double-Bit Quantization

In the previous section, it elaborates that binary embedding methods greatly reduce the discriminability of original signatures. To achieve higher accuracy, most of related works concentrate on improving the performance of projection functions g_k. Instead, we propose a DBQ method to assign double-bit to each dimension of the intermediated data. The steps of DBQ are summarized below:

1) Signature projection and normalization. For a given signature *x* with *K* dimensions, we use a multidimensional projection function $g(s) = [g_k(s), k = 1 \ldots K]'$, such as locality sensitive hashing [27], principal component analysis [35], spectral hashing [38], PCA-RR [35] and iterative quantization [40], to map original signature *x* to real-valued vector $g(x)$ (the intermediated data). To enhance the efficiency of comparisons, the vector is normalized by its l_1 norms $\mathcal{N}$ for each dimension and the normalized intermediated data $l(x) = [l_k(x), k = 1 \ldots K]'$ is obtained by

$$l_i(s) = \mathcal{N}[g_i(s)]. \quad i = 1 \ldots K \quad (4)$$

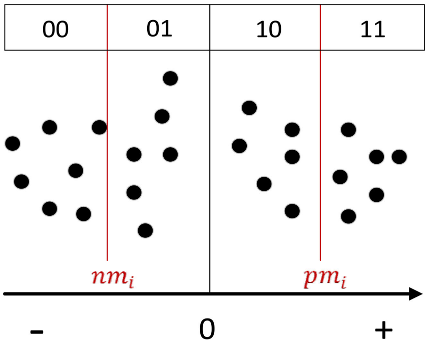


Fig. 3. Encoding signature. Each dimension is divided into four parts by the sign and two medians.

2) Data partition. We divide the intermediate data of each dimension into two categories according to the sign of the corresponding element, after which we get the medians of both categories for all the dimensions. The medians of the negative and positive parts in dimension *i* are represented symbolically by nm_i and pm_i, respectively. Based on the sign and two medians, each element of the intermediated vectors can be divided into four categories, as illustrated in Fig. 3. Although it is simple, the partition scheme leads to competitive results on a variety of binary embedding methods.

3) Binary quantization. By original quantization function, the positional relations of elements in each dimension have only two cases: either on the same side or the opposite side. To increase the discrimination, we quantize each dimension into double bits. In this way, the quantization method may adapt well to the four relations of the elements in each dimension, as shown in Fig. 3. For the i_{th} dimension, the DBQ function is defined as:

$$DBQ_i(s) = \begin{cases} 11, \text{ if } l_i(s) \geq pm_i \\ 10, \text{ if } l_i(s) \geq 0 \text{ and } l_i(s) < pm_i \\ 01, \text{ if } l_i(s) \langle 0 \text{ and } l_i(s) \rangle\, nm_i \\ 00, \text{ if } l_i(s) \leq nm_i \end{cases} \quad (5)$$

Since intermediate data preserves good approximation to the similarity of original signatures, it has a high probability to map $g_i(x)$ and $g_i(y)$ to the same category, if *x* is the nearest neighbor of *y*. Conversely, if signature *x* and *y* are far from each other, $g_i(x)$ and $g_i(y)$ are more likely to be mapped far apart. Thus, the quantization scheme can naturally preserve the similarity between two signatures.

As the intermediated data are generate by existing hash functions, DBQ does not violate the design objectives of the data-driven learning of hash functions. It also attempts to minimize the Hamming distance between the similar training samples while satisfying some constraints, such as bits should be evenly distributed and bits must be independent.

TABLE I
WEIGHTED HAMMING DISTANCE BETWEEN EACH DIMENSION

	00	01	10	11
00	0	1	2	3
01	1	0	1	2
10	2	1	0	1
11	3	2	1	0

For binary codes generated from DBQ, we cannot directly calculate the hamming distance between them. There are two reasons. Firstly, DBQ partitions each dimension of the intermediate data into four parts. Thus, the quantized signatures have four kinds of spatial relationship which can be represented by 4 kinds of distances (0, 1, 2 and 3). However, the hamming distance between 2-bit binary codes only has three possible values (0, 1 and 2). Secondly, XOR operation cannot describe the distance between DBQ binary codes accurately. For example, the distance between 01 and 10 is 1, but the hamming distance between them is 2. Likewise, the distance between 00 and 11 is 3, while the hamming distance is 2. Therefore, we propose a novel weighted hamming distance to satisfy the distance computation of DBQ binary codes, as shown in Table I.

For a given query, we first build several look-up tables, and then the weighted hamming distance is calculated by searching the pre-computed look-up tables. Note that, one dimension refers to 2-bit binary code in the following description for clear expression.

Assume *x*, *q* are the intermediate data of reference signature and query signature respectively. To calculate the weighted hamming distance between *x* and *q* online, we should sum up the distance of each dimension composed of 2 bits. Let $d_k(DBQ_k(x), DBQ_k(q))$ represents the weighted hamming distance between the *k*-th dimension of *x* and *q* which is demonstrated as follows:

Online, for the given query *q*, the following query-dependent values are pre-computed and stored in look-up tables:

$$\beta_k^{x,q} = d_k\left(DBQ_k\left(x\right), DBQ_k\left(q\right)\right). \tag{6}$$

Assume d_{WH} stands for the weighted hamming distance between *x* and *q*. By definition, we have

$$d_{WH}\left(x,q\right) = \sum_k \beta_k^{x,q}. \tag{7}$$

The cost of computing the values β is negligible with comparison to that of computing $d_{WH}(x,q)$ for a large number of reference signatures *x*. The sum (7) can be computed very efficiently by grouping the dimensions. In our implementation, we subdivide a vector into blocks of four dimensions (8-bit). Assuming that the number of dimensions *K* is a multiple of 4, to simplify the notation, we get:

$$d_{WH}\left(x,q\right) = \sum_{k=0}^{K/4-1}\sum_{j=1}^{j=4}\beta_{4k+j}^{x,q}. \tag{8}$$

Because the binary subvector $[DQ_{4k}(x), DQ_{4k+1}(x)$ $DQ_{4k+2}(x), DQ_{4k+3}(x)]$, fits in one byte, each sum $\sum_{j=1}^{j=4}\beta_{4k+j}^{x,q}$ can only take 256 possible values. We can precompute these 256 values and store them in a look-up table. In total, we need to calculate *K*/4 tables for the computation.

Generally, the *K*/4 look-up tables will be cached as they only need little storage space. Therefore, the weighted hamming distance can be simply calculated by adding values from each look-up table together, rather than computing hamming distance for each reference binary code via XOR operation.

C. Double-Bit Index Hashing

The structure of the double-bit index hashing is consistent with MIH [32]. In order to improve the retrieval efficiency, the binary codes are divided into several non-overlapping substrings and multiple hash tables are built for each of them. For a given query, it is also divided into several substrings in the same way. Then each query substring gets the nearest neighbors from the corresponding hash table. Next, we take these nearest neighbors of substrings as candidates and compare the entire binary codes with the query to remove any non-neighbors. Finally, the obtained results are the nearest neighbors of the query.

As mentioned in Section II, MIH gets the nearest neighbors by increasing the hamming distance and examining the corresponding hash buckets. Given a query *q*, the address of hash bucket is calculated by *q* XOR mask, where the number of 1 in mask represents the hamming distance. For example, assume *q* equals to 00011011 and the expected hamming distance is 3. Then we get one of the mask valued 00000111. The result of *q* XOR mask is 00011100, which has the hamming distance of 3 for *q*.

However, the XOR operation is not suitable for the proposed DBQ binary codes when performing NN search. Therefore, a method is proposed to apply DBQ codes to binary code indexing based on MIH, with the structure of MIH remains unchanged. As shown in Table I, there are four kinds of binary codes in each dimension, namely 00, 01, 10 and 11. They are corresponding to 0, 1, 2 and 3 in decimal, respectively, and the range of the distance for each kind of dimension is diverse. For example, 00 can be changed to 01, 10 and 11 by $+1$, $+2$ and $+3$ respectively. For 10, the changes include $+1$, -1 and -2. Since the different distances are equivalent to the weights in dimension, the increase or decrease in value is considered as weighted hamming distance (WHD). Therefore, in the process of increasing the weighted hamming distance for NN search, each kind of dimension has different cases, rather than a simple 0/1 bit flip. So, we will discuss different situations separately. Specific steps are as follows:

1) Resize the *mask*. The number of 1 in *mask* (in binary) equals to the WHD. Further, the number of 1 in a dimension of *mask* represents the WHD of the corresponding dimension. Assume that c_1 is the number of 00 and 11, and c_2 is the number of 10 and 01. As the range of WHD of 11 and 00 is 3, and that of 01 and 10 is 2, the length of the *mask* is set to be $3 \times \mathrm{c}_1 + 2 \times \mathrm{c}_2$, where $\mathrm{c}_1 + \mathrm{c}_2 = b/2$. For example, as shown in Table II, the changes of the *mask* 00011011 are presented, where the WHD ranges from 0 to 10.

TABLE II
THE CHANGES OF MASK FOR 8-BIT BINARY CODE 00011011 WHEN INCREASING THE WEIGHTED HAMMING DISTANCE

binary code / WHD	00	01	10	11	number
	one of the cases				of cases
r=0	000	00	00	000	1
r=1	000	00	00	001	C_{10}^{1}
r=2	000	01	10	000	C_{10}^{2}
r=3	011	01	00	000	C_{10}^{3}
…	……				
r=10	111	11	11	111	1

TABLE III
THE CHANGES OF CORRESPONDING DIMENSION OF *IN* AND *RN* WHEN BINARY CODE IS 00 OR 11. '+' REPRESENTS THE CHANGE IN *IN;* '-' REPRESENTS THE CHANGE IN *RN*

mask / value	000	001	010	011	100	101	110	111
00	0	+1	+1	+2	+1	+2	+2	+3
11	0	-1	-1	-2	-1	-2	-2	-3

TABLE IV
THE CHANGES OF CORRESPONDING DIMENSION OF *IN* AND *RN* WHEN BINARY CODE IS 01 OR 10. '+' REPRESENTS THE CHANGE IN *IN;* '-' REPRESENTS THE CHANGE IN *RN*

mask / value	00	01	10	11
01	0	+1	-1	+2
10	0	+1	-1	-2

2) Transform the *mask*. When the WHD increases, the transformation of *mask* for NN search is the same as MIH except that the length of the *mask* is different. So, when the WHD increases to *R*, the total number of the cases of *mask* is $C^{R}_{3\times c1+2\times c2}$. In each case, there exist different bit-wise flips in *mask*. We first initialize two variables *IN* and *RN* to 0, which represent the numbers to be added and subtracted to the query, respectively. Note that the two variables have the same number of bits as the query, as every two bits of them represents the WHD of the corresponding dimension.

Table III and Table IV are presented to detail the changes of *IN* and *RN* according to different *mask* and binary codes. For 00 (see Table III), only *IN* can be used, meaning a certain number can be added to the dimension. While, 11 is just on the contrary. As for 01 (see Table IV), the change ranges from -1 to $+\,2$, which indicates *IN* can be set as 01 and 10 or *RN* can be set as 01 in corresponding dimension. However, when the WHD is 1, *IN* and *RN* can both be utilized. In this case, we use *mask* to determine which variable to change. It requires that when the dimension of *mask* equals 01, the corresponding dimension of *IN* is set as 01, while *RN* is set as 01 when *mask* is 10. The dimension of 10 works the same way. With the simple approach, it guarantees that each transformation of *mask* corresponds to only one situation, and takes into account all the cases at the same time.

3) Examine the hash buckets. After the previous steps, the address of the hash bucket to be examined is

$$\text{address} = \text{query} + IN - RN \tag{9}$$

Subsequent search approaches are consistent with the MIH.

Here is an example, given the query 01001011, which is 8-bit code (4-dimension), and the length of the *mask* is set to be 10-bit long. If the expected WHD is 4, the *mask* should contain four bits valued 1. Assume the *mask* is 0010110100, for the first dimension (from right to left), the corresponding dimension of *mask* does not contain any 1, so there is no change in *IN* or *RN*. For the second dimension of the query 00, the *mask* (containing 3 bits in this dimension) has two bits valued 1, so $IN = IN|00100000$. For the third dimension of the query 10, the corresponding *mask* has one bit valued 1, then $RN = RN|00000100$ For the fourth dimension 11 of the query, there is one bit valued 1, so $RN = RN|00000001$. As a result, the *IN* equals to 00100000 and *RN* equals to 00000101. Thus, the address of hash bucket to be examined is 01100110, which has the WHD of 4 for the query.

This method takes into account all cases of the WHD, and each case appears only once. To fit DBQ binary codes, it considers different combinations of *mask* and binary code to get the address of hash buckets to be examined. As revealed in the experiment results, DBIH remarkably increases the search efficiency compared to the linear search.

D. Weighted Distance Measurement

Binary codes are storage efficient and fast to compute. Millions of binary codes can be compared to a query in less than a second. However, the discrimination of the binary codes is still limited, even though DBQ is employed. Thus, the weighted distance measurement is proposed to further increase the retrieval accuracy.

As stated before, Albert *et al.* [44] propose an algorithm for asymmetric distance based on expected value, which is defined in equation (3). In this paper, we utilize the concept of [44] and propose the WDM to satisfy DBQ binary codes. The calculation of WDM computes the distance between uncompressed data and the DBQ binary code, which is equivalent to assigning different weights to each dimension according to the position of query. A major benefit of the WDM is that it can achieve higher accuracy, as it takes advantage of the precise information of the query.

For a given query, we firstly get the top-*k* closest signatures in hamming space from DBIH. Then, these results are considered as candidates to be re-ranked through WDM. Note that, we replace h_k with the proposed DBQ_k in equation (3). To get expected value for each dimension, a set of signatures $\boldsymbol{S}\{s_i, i = 1 \ldots N\}$ is firstly drawn randomly from Ω. For each dimension *k*, we partition $\boldsymbol{S}$ into four clusters $\boldsymbol{S}_k^{cu}, cu = 00, 01, 10, 11$. Each cluster contains signatures *s* such that $DBQ_k(s) = cu$. Then, the expected value c_k is computed for each subsets offline:

$$c_k^{cu} = \frac{1}{|\boldsymbol{S}_k^{cu}|} \sum_{v \in \boldsymbol{S}_k^{cu}} g_k\,(v). \tag{10}$$

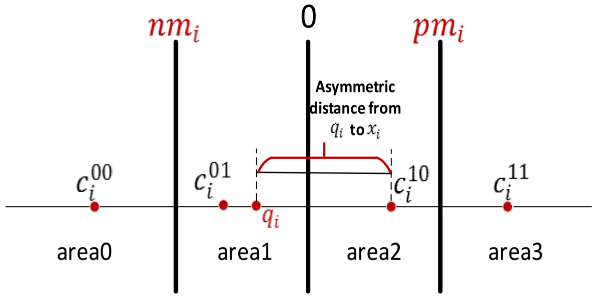


Fig. 4. Asymmetric distance.

For a given query *x*, the distance (see Fig. 4) is calculated between uncompressed query and the expected value of corresponding dimension of DBQ binary code online, which is defined as:

$$d_E\left(x\right) = \sqrt{\sum_k \left(g_k\left(x\right) - c_k^{DBQ_k(x)}\right)^2}. \tag{11}$$

It shows that the WDM takes full advantages of the non-binarized signature, which keeps the information of the original. For example, for *i*-th dimension of a given query *q* and reference signature *x*, $g_i(q)$ locates in area1 and $DBQ_i(x)$ is mapped to area2. The weighted distance between them is the Euclidean distance between $g_i(q)$ and the expected value of all the intermediated data mapped to area2 in dimension *i*. Along with the benefit of DBQ, the discriminability can be further improved, due to different signatures possessing four possible position relations through WDM.

In system implementation, to compare the performance objectively, we select top 100 nearest neighbors in Hamming space as the candidates to evaluate the effect of WDM. Even though extra floating-point multiplication is involved in the process, it has negligible time consumption, as we calculate the weighted distance only for limited number of signatures. In addition, the calculation can be done by pre-computed look-up tables just as the double-bit quantization.

IV. Experiments

In this section, we show the performance of our proposed method. We first introduce three famous datasets and evaluation metrics used in the experiments. Then intensive comparisons between the framework of DBQ-IH and traditional methods are given.

A. Dataset and Evaluation Metrics

Three benchmark datasets are used to evaluate DBQ and WDM, including 1M SIFT (128-d) and 1M GIST (960-d) signatures from BIGANN dataset [11] and GIST (320-d) signatures from Caltech101 dataset [33]. For DBIH evaluation, we use 1B SIFT (128-d) features from BIGANN dataset, which consists of three vector subsets: training set, database set and query set.

For experimental evaluation of DBQ, the precision and recall are employed to measure the performance of our method. The WDM is measured by precision. The precision and recall are defined as follows:

$$precision = \frac{count(relevant \cap retrieved)}{retrieved} \tag{12}$$

$$recall = \frac{count(relevant \cap retrieved)}{relevant}, \tag{13}$$

where *retrieved* is the NN search results of our method, and *relevant* is the linear search results in Euclidean space. We compare the recall@10 and precision@1 of several similarity-preserving binary embedding methods, namely, locality sensitive hashing (LSH) [27], principal component analysis (PCA) [35], spectral hashing (SH) [38], PCA-RR [35], and iterative quantization (ITQ) [40], with DBQ and WDM.

For DBIH evaluation, the method is verified by query speed. The run-time of proposed DBIH will be compared with linear search, using the DBQ binary codes generated from LSH.

B. Results

Our proposed framework is a binary based large-scale nearest neighbor search system. After the signatures are obtained by various methods, binary embedding methods are utilized to convert the floating-point signatures to binary codes, of which the double-bit quantization method is employed. Next, the double-bit index hashing creates a multiple index hashing tables to obtain the nearest neighbors in the Hamming space. We use the weighted distance measurement to re-rank the results from DBIH for the final nearest neighbors. Therefore, our experiments can be divided into the following three parts:

- Double-bit quantization. Comparing the retrieval precision and recall of the traditional binary embedding methods with the DBQ binary codes, on the baseline of the nearest neighbors in Euclidean space.
- Double-bit index hashing. Comparing the query speed of the linear scan with the DBIH in the dataset composed of DBQ binary codes.
- Weighted distance measurement. Comparing the retrieval precision of the results from DBIH with re-ranked results through WDM, on the same baseline of double-bit quantization.

As we can see from the summary, the three-step experiments are progressive layers in relationship. Since the experiments of DBQ and WDM use the same measurements and baseline, the settings for them are introduced together.

1) Double-Bit Quantization:

- Double-bit quantization *VS* original method

Each experiment has 1000 queries, in which precision and recall are used as metrics. For the different binary embedding methods in three different datasets, the results using DBQ are better than original binary embedding methods.

To get the double-bit binary codes, the signatures in training set are mapped to the intermediate data, and the medians are obtained according to the sign of each dimension. Then we transform both reference signatures and query signatures to intermediate data in the same way as training set. Next, the data are quantized into double-bit binary codes through DBQ. Finally,

TABLE V

SUMMARY OF RESULTS (PERCENT) FOR BIGANN 1M SIFT DATASET. CODES ARE 32, 64, 128 OR 256 BITS LONG, OBTAINED BY ITQ, RR, SH, LSH OR PCA. P@1 MEANS THE PRECISION@1, AND R@10 STANDS FOR THE RECALL@10. SB REPRESENTS THE SINGLE-BIT QUANTIZATION AND DB MEANS OUR DOUBLE-BIT QUANTIZATION

mapping	32				64				128				256			
	P@1		R@10		P@1		R@10		P@1		R@10		P@1		R@10	
	SB	DB	SB	DB	SB	DB	SB	DB	SB	DB	SB	DB	SB	DB	SB	DB
ITQ	7.8	**10.1**	4.2	**5.3**	15.4	**21.8**	8.9	**15.3**	25.1	**37.8**	18.2	**29.3**	-	**42.8**	-	**39.6**
RR	8.2	**8.5**	4.4	**4.5**	15.8	**18.1**	9.8	**12.6**	26.3	**38.5**	18.9	**30.3**	-	**49.9**	-	**42.4**
SH	4.2	3.8	3.5	**4.9**	9.8	**11.7**	10.2	**17.5**	16.7	**24.3**	17.4	**28.9**	28.7	**34.2**	25.6	**38.8**
LSH	2.2	**2.3**	1.1	**1.2**	5.3	**5.5**	5.6	**5.8**	10.2	**13.3**	11.5	**13.2**	20.2	**22.8**	22.5	**28.6**
PCA	4.8	**4.5**	6.8	6.4	10.5	**11.6**	11.2	**17.5**	17.6	**27.6**	17.2	**22.2**	-	**28.2**	-	**28.9**

TABLE VI

SUMMARY OF RESULTS (PERCENT) FOR CALTECH101 GIST DATASET. CODES ARE 64, 128, 256 OR 320 BITS LONG, OBTAINED BY ITQ, RR, SH, LSH OR PCA. P@1 MEANS THE PRECISION@1, AND R@10 STANDS FOR THE RECALL@10. SB REPRESENTS THE SINGLE-BIT QUANTIZATION AND DB MEANS OUR DOUBLE-BIT QUANTIZATION

mapping	64				128				256				320			
	P@1		R@10		P@1		R@10		P@1		R@10		P@1		R@10	
	SB	DB	SB	DB	SB	DB	SB	DB	SB	DB	SB	DB	SB	DB	SB	DB
ITQ	8.2	**11.5**	10.1	**18.2**	14.2	**22.4**	14.5	**27.6**	18.4	**29.6**	19.1	**34.1**	28.9	**32.6**	19.5	**35.8**
RR	9.8	**12.2**	10.2	**18.7**	13.4	**18.7**	13.2	**27.7**	18.9	**27.6**	18.7	**33.7**	19.8	**30.2**	19.8	**36.3**
SH	3.2	**5.3**	7.8	**12.2**	7.8	**10.2**	10.7	**18.9**	13.2	**22.1**	12.5	**23.6**	16.7	**23.1**	12.7	**24.3**
LSH	7.2	**7.2**	6.3	**6.5**	11.7	**12.5**	10.8	**12.1**	16.3	**18.6**	17.8	**20.2**	17.6	**20.1**	18.2	**22.8**
PCA	8.5	**12.7**	7.6	**12.3**	9.9	**15.4**	7.7	**14.7**	10.1	**14.3**	8.6	**11.3**	10.0	**13.2**	7.6	**8.9**

TABLE VII

SUMMARY OF RESULTS (PERCENT) FOR BIGANN 1M GIST DATASET. CODES ARE 64, 128, 256 OR 512 BITS LONG, OBTAINED BY ITQ, RR, SH OR LSH. P@1 MEANS THE PRECISION@1, AND R@10 STANDS FOR THE RECALL@10. SB REPRESENTS THE SINGLE-BIT QUANTIZATION AND DB MEANS OUR DOUBLE-BIT QUANTIZATION

mapping	64				128				256				512			
	P@1		R@10		P@1		R@10		P@1		R@10		P@1		R@10	
	SB	DB	SB	DB	SB	DB	SB	DB	SB	DB	SB	DB	SB	DB	SB	DB
ITQ	6.5	**7.8**	8.9	**12.1**	6.6	**13.5**	8.8	**19.2**	8.6	**19.6**	10.6	**25.9**	10.1	**26.3**	12.1	**31.5**
RR	4.0	**6.9**	6.5	**11.6**	6.5	**13.5**	8.4	**19.0**	8.0	**20.2**	10.7	**25.9**	10.3	**19.6**	12.4	**25.9**
SH	1.2	**5.7**	2.3	**3.5**	2.4	**6.3**	2.9	**9.1**	2.8	**8.1**	4.6	**10.7**	4.5	**10.0**	6.0	**11.9**
LSH	0.8	**1.2**	0.9	**1.3**	2.4	**4.6**	3.1	**6.9**	5.2	**10.1**	6.8	**13.9**	7.2	**16.6**	8.7	**21.6**

the weighted hamming distance is calculated for each binary code.

Table V, VI and VII show the comparison results on three datasets of original binary embedding algorithms and those with DBQ. The results demonstrate that DBQ consistently improves the retrieval accuracy over the original binary embedding algorithms and is independent of the datasets, the descriptors and the binary embedding methods. Here are two examples: on Caltech101 (Table VI), we observe a relative improvement on recall of 80.2% at 256-bit when using RR-PCA; on BIGANN, when SH is employed we can observe a relative improvement of precision of 42.3% (Table V) at 128 bits. The comparison results show that all methods have obvious improvement with DBQ. This can be explained by two primary factors.

On one hand, double-bit quantization preserves more information of original signatures. The average discrimination per bit declines as the dimensionality increases. Here is an example, for ITQ, when the dimensionality k is 128, the precision is 14.2% on Calteach101. Given the same circumstances except for the dimension being $2k$, the precision is 16.9% which only has 19% relative improvement. In theory, $2k$-bit binary code with DBQ doubles the retrieval accuracy of k-bit binary code with one-bit quantization. That is to say, when the dimension of binary codes is both $2k$, the discrimination of DBQ significantly outperforms one-bit quantization.

On the other hand, weighted hamming distance has more distinguishing ability. When one-bit quantization is used, the hamming distance of each two-bit code has three possible values (0, 1 and 2). But it has four possibilities (0, 1, 2 and 3) when using DBQ. Thus, weighted hamming distance has a wider range than original method. For example, assume the number of bit is 64, and then the range of hamming distance is 0 to 64. However, the range of weighted hamming distance is 0 to 128, which is twice of the former. As a consequence, DBQ obtains stronger discrimination than traditional binary embedding methods.

- Double-bit quantization *VS* Manhattan hashing [47]

To verify the effectiveness of DBQ, Manhattan hashing (MH) is compared. The basic idea of MH is to encode each projected

TABLE VIII
COMPARISON BETWEEN DOUBLE BIT QUANTIZATION AND MANHATTAN HASHING [47] (PERCENT) FOR BIGANN 1M SIFT DATASETS. CODES ARE 32, 64, 128 OR 256 BITS LONG, OBTAINED BY ITQ, RR, SH, LSH OR PCA. P@1 MEANS THE PRECISION@1, AND R@10 STANDS FOR THE RECALL@10. MH REPRESENTS MANHATTAN HASHING AND DB MEANS OUR DOUBLE-BIT QUANTIZATION

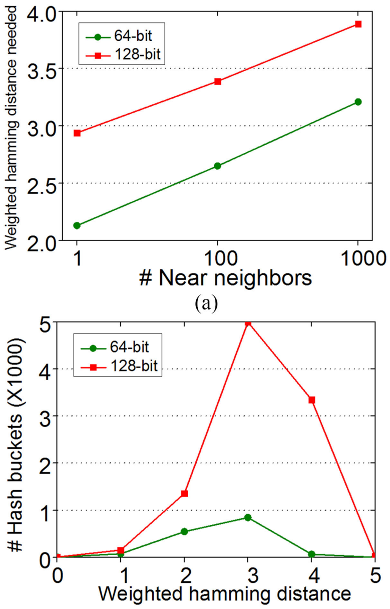
mapping	32				64				128				256			
	P@1		R@10		P@1		R@10		P@1		R@10		P@1		R@10	
	MH	DB	MH	DB	MH	DB	MH	DB	MH	DB	MH	DB	MH	DB	MH	DB
ITQ	1.15	**10.1**	4.11	**5.3**	9.4	**21.8**	14.2	**15.3**	22.9	**37.8**	27.5	**29.3**	33.1	**42.8**	38.8	**39.6**
RR	0.5	**8.5**	1.7	**4.5**	3.6	**18.1**	5.31	**12.6**	9.0	**38.5**	8.61	**30.3**	13.2	**49.9**	13.0	**42.4**
SH	2.7	**3.8**	4.86	**4.9**	11.4	**11.7**	16.2	**17.5**	21.3	**24.3**	29.1	28.9	33.8	**34.2**	36.7	**38.8**
LSH	1.3	**2.3**	1.93	1.2	5.8	5.5	6.96	5.8	14.8	13.3	15.8	13.2	25.2	22.8	29.9	28.6
PCA	2.4	**4.5**	4.94	**6.4**	12.1	11.6	15.6	**17.5**	22.7	**27.6**	26.8	22.2	25.6	**28.2**	26.4	**28.9**

dimension with multiple bits of natural binary code, based on which the Manhattan distance between points in the hamming space is calculated. In order to maintain consistency of the experiments, the number of bit per dimension is set to be 2. The dataset we use is BIGANN SIFT 1M. The experiment has 1000 queries, in which precision and recall are adopted as metrics. As shown in Table VIII, DBQ outperforms MH in most of the cases expect when the hashing method is LSH due to the randomness.

Note that NPQ has better performance than MH [49]. To our knowledge, DBQ has the best accuracy on the three datasets at present. For the reason of paper length, wo do not compare our method with NPQ.

2) Double-Bit Index Hashing: Each experiment involves 10000 queries, and we report the average running time. When the number of bits is 64, we divide the binary codes into 4 segments. The 128-bit long binary codes are divided to 8 segments. The results illustrate that DBIH is remarkably faster compared to the linear scan. Note that the linear scan speed relies heavily on the memory cache. If there is less cache, linear scan will be much slower [32].

DBIH only needs small number of hash buckets to be examined. Fig. 5(a) displays the average search radius required for1000 queries on a dataset of 10^8 SIFT descriptors. As we can see, the search radiuses are concentrated in the range of 2 to 4. Fig. 5(b) shows the relationship between the number of examined hash buckets and the hamming distance on the same dataset. For linear scan, each signature is equivalent to one hash bucket. So it elaborates that DBIH requires a much smaller number of hash buckets than linear scan. Thus, DBIH is supposed to perform more efficient NN search than linear scan.



(a)

(b)

Fig. 5. (a) Shows the average search radius required according to the number of nearest neighbor expected. (b) Elaborates the relationship of hamming distance and the number of hash buckets to be examined.

Fig. 6 and Fig. 7 show the query speed of linear scan and DBIH, for different *k*-NN problems and two different numbers of bits: 64-bit and 128-bit. Although linear scan for binary code is fast enough, DBIH performs several times faster. For example, DBIH resolves the exact 100-NN in a 2×10^8 128-bit dataset with about 0.71 s, 5 times faster than the linear scan (see Fig. 7). Among them, 1-NN and 10-NN performance are even more impressive. With 64-bit LSH codes, DBIH executes the 1-NN search task over 30 times faster than the linear scan. The runtime of the linear scan is independent of the expected number of neighbors. However, the query time of DBIH depends on this factor. In particular, as the number of expected neighbors increases, the hamming radius of the NN search also increases, meaning the longer query time for DBIH. Moreover, we notice that the query speed of DBIH is sub-linear as the size of the data set grows, since the query hamming radius is not directly dependent on the size of the data set, in the case of a fixed number of neighbors.

3) Weighted Distance Measurement: Even though the WDM is used to re-rank the results from DBIH, we use the datasets obtained by the experiments of DBQ for more convincing verification. Each experiment has 1000 queries, in which precision is used as a metric. Because we only re-rank the results, the recall remains unchanged. For the different binary embedding methods in three different datasets, the re-ranked results are better than the results directly obtained.

Figs. 8, 9 and 10 indicate the precision before and after the use of WDM for different binary embedding methods in different datasets. Even though DBQ has improved the precision remarkably, the WDM can still further enhance the retrieval accuracy. As we can see in CalTech101 (Fig. 9), the WDM doubles the retrieval accuracy for most binary embedding methods. In SIFT and GIST signature datasets from BIGANN, the performance

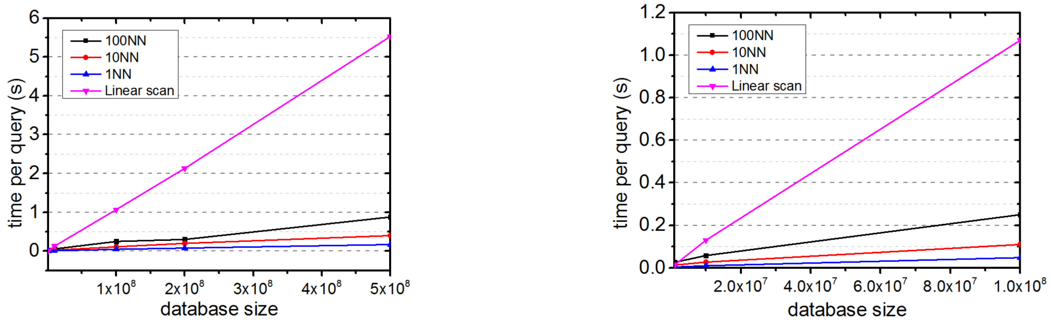


Fig. 6. Run-times per query for double-bit index hashing with 1, 10 and 100 nearest neighbors, and a linear scan baseline on 5×10^8 64-bit binary codes generated by LSH from SIFT in BIGANN.

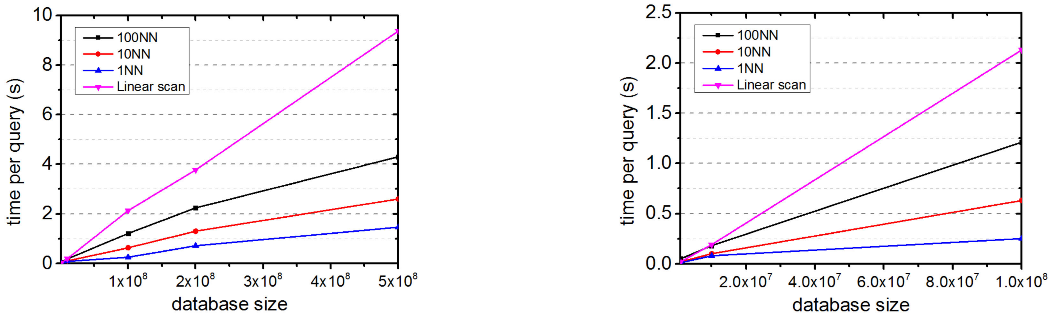


Fig. 7. Run-times per query for double-bit index hashing with 1, 10 and 100 nearest neighbors, and a linear scan baseline on 5×10^8 128-bit binary codes generated by LSH from SIFT in BIGANN.

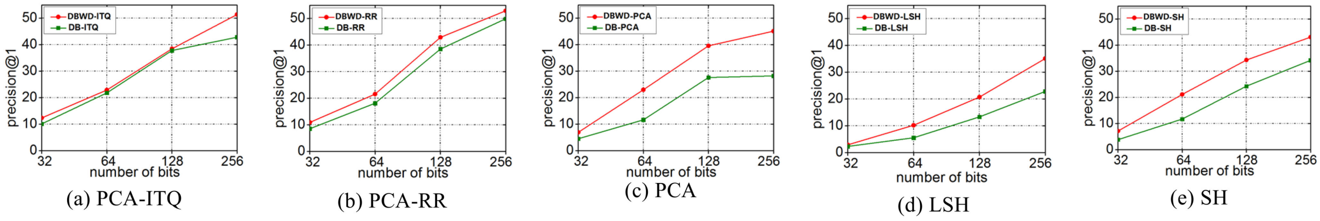

(a) PCA-ITQ (b) PCA-RR (c) PCA (d) LSH (e) SH

Fig. 8. Influence to precision of the weighted distance measurement on the BIGANN dataset using 128-d sift descriptors through double-bit quantization and different binary embedding methods, including ITQ, RR, PCA, LSH and SH. From left to right: 32-bit, 64-bit, 128-bit and 256-bit codes. DB represents the double-bit quantization and DBWD means the double-bit quantization with weighted distance measurement.

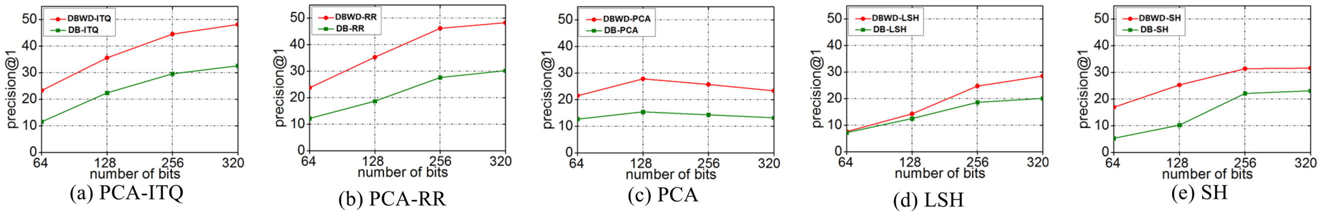
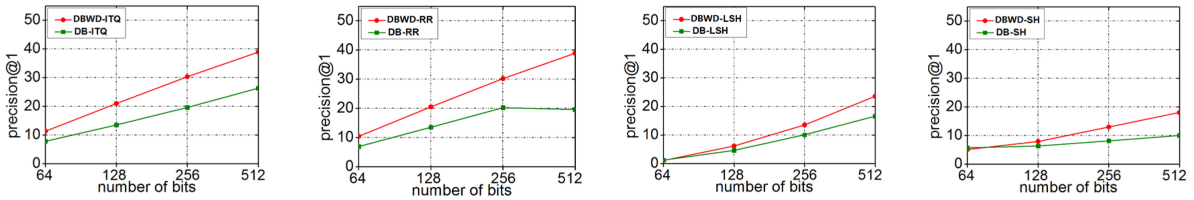

(a) PCA-ITQ (b) PCA-RR (c) PCA (d) LSH (e) SH

Fig. 9. Influence to precision of the weighted distance measurement on the CalTeach101 dataset using 320-d gist descriptors through double-bit quantization and different binary embedding methods, including ITQ, RR, PCA, LSH and SH. From left to right: 64-bit, 128-bit, 256-bit and 320-bit codes. DB represents the double-bit quantization and DBWD means the double-bit quantization with weighted distance measurement.

is also excellent: in Fig. 8(c), the query accuracy is improved by an average of 58.3%. We have noticed that when the number of bits is small, the effect of WDM is not obvious. As the binary codes are obtained from DBQ, every two bits represents just one dimension. However, the WDM is calculated by each dimension. So the benefits of WDM will be limited when the number of bits is small.

In summary, the empirical results on large-scale binary code databases demonstrate the superiority of our framework over existing approaches in terms of both retrieval accuracy and query efficiency. We observe an absolute improvement on precision of 10%∼25% compared to traditional binary embedding methods in most cases. Meanwhile, the query speed of our method is increased over 30 times than linear scan.



(a) PCA-ITQ (b) PCA-RR (c) LSH (d) SH

Fig. 10. Influence to precision of the weighted distance measurement on the BIGANN dataset using 960-d gist descriptors through double-bit quantization and different binary embedding methods, including ITQ, RR, LSH and SH. From left to right: 64-bit, 128-bit, 256-bit and 512-bit codes. DB represents the double-bit quantization and DBWD means the double-bit quantization with weighted distance measurement.

Note that not all dimensions are equally informative and important and quite likely that some might not require two bits. In the design of DBQ, we have considered the scheme of assigning different bits for each dimension. Actually, we have assigned different bits for each dimension just on the basis of the covariance of each dimension and got promising results. However, we did not give the comparison results for the reason of structure arrangement.

V. Conclusion

In this paper, we propose a novel framework double-bit quantization and index hashing for fast large-scale nearest neighbor search. On one hand, it takes advantages of both compressed data and real-valued signatures for higher accuracy. On the other hand, DBQ-IH utilizes the benefit of double-bit index hashing to accelerate large-scale NN search.

To improve retrieval accuracy, we quantize each dimension of intermediate data in the dataset into double-bit binary code. Then double-bit index hashing is employed to organize the DBQ binary codes for fast NN search. Finally, the results from DBIH are re-ranked by weighted distance measurement to further improve the retrieval accuracy.

Experimental results illustrate DBQ-IH can achieve an absolute improvement on precision of 10%$\sim$25% compared to the original binary embedding methods. It indicates that double-bit quantization and weighted distance measurement perform superior results by assigning more weight to the signatures close to query both in Hamming space and Euclidean space. At the same time, our framework accelerates the query speed by over 30 times compared to linear scan. We believe the proposed DBQ-IH can improve the retrieval accuracy and efficiency of many nearest neighbor search applications. As not all dimensions are equally informative and important and quite likely that some might not require two bits, in the future we will study assigning different bits for each dimension and designing consequent indexing structure.

References

[1] J. Song *et al.*, “A distance computation free search scheme for binary code databases,” *IEEE Trans. Multimedia*, vol. 18, no. 3, pp. 484–495, Mar. 2016.

[2] H. Wang, L. Feng, J. Zhang, and Y. Liu, “Semantic discriminative metric learning for image similarity measurement,” *IEEE Trans. Multimedia*, vol. 18, no. 8, pp. 1579–1589, Aug. 2016.

[3] Y.-G. Jiang, J. Wang, X. Xue, and S.-F. Chang, “Query-adaptive image search with hash codes,” *IEEE Trans. Multimedia*, vol. 15, no. 2, pp. 442–453, Feb. 2013.

[4] S. Megrhi, W. Souidène, and A. Beghdadi, “Spatio-temporal SURF for human action recognition,” in *Proc. Adv. Multimedia Inf. Process.*, 2013, pp. 375–385.

[5] H. Xie *et al.*, “Efficient feature detection and effective post-verification for large scale near-duplicate image search,” *IEEE Trans. Multimedia*, vol. 13, no. 6, pp. 1319–1332, Dec. 2011.

[6] Y.-G. Jiang, J. Wang, Q. Wang, W. Liu, and C.-W. Ngo, “Hierarchical visualization of video search results for topic-based browsing,” *IEEE Trans. Multimedia*, vol. 18, no. 11, pp. 2161–2170, Nov. 2016.

[7] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov, “Scalable object detection using deep neural networks,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2155–2162.

[8] J. Revaud, M. Douze, C. Schmid, and H. Jégou, “Event retrieval in large video collections with circulant temporal encoding,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2013, pp. 2459–2466.

[9] Y.-G. Jiang, M. Li, X. Wang, W. Liu, and X.-S. Hua, “DeepProduct: Mobile product search with portable deep features,” *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 14, no. 2, pp. 50:1–50:18, 2018.

[10] H. Xie *et al.*, “Effective and efficient image copy detection based on GPU,” in *Proc. Eur. Conf. Comput. Vis.*, 2010, pp. 338–349.

[11] H. Jegou, R. Tavenard, M. Douze, and L. Amsaleg, “Searching in one billion vectors: Re-rank with source coding,” in *Proc. Int. Conf. Acoust., Speech, Signal Process.*, 2011, pp. 861–864.

[12] H. Xie, D. Yang, N. Sun, Z. Chen, and Y. Zhang, “Automated pulmonary nodule detection in CT images using deep convolutional neural networks,” *Pattern Recognit.*, vol. 85, pp. 109–119, 2019. [Online]. Available: https://doi.org/10.1016/j.patcog.2018.07.031

[13] L. Herranz, S. Jiang, and X. Li, “Scene recognition with CNNs: Objects, scales and dataset bias,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 571–579.

[14] A. A. Cruzroa *et al.*, “A deep learning architecture for image representation, visual interpretability and automated basal-cell carcinoma cancer detection,” in *Proc. Int. Conf. Med. Image Comput. Comput.-Assisted Intervention*, 2013, pp. 403–410.

[15] C. Yan *et al.*, “Supervised hash coding with deep neural network for environment perception of intelligent vehicles,” *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 1, pp. 284–295, Jan. 2018.

[16] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *Int. J. Comput. Vis.*, vol. 60, pp. 91–110, 2004.

[17] H. Xie *et al.*, “Pairwise weak geometric consistency for large scale image search,” in *Proc. 1st ACM Int. Conf. Multimedia Retrieval*, 2011, pp. 1–8.

[18] I. Yahiaoui, N. Herve, and N. Boujemaa, “Shape-based image retrieval in botanical collections,” in *Proc. Adv. Multimedia Inf. Process.*, 2006, pp. 357–364.

[19] A. C. Murillo and J. Kosecka, “Experiments in place recognition using gist panoramas,” in *Proc. IEEE 12th Int. Conf. Comput. Vis. Workshops*, 2009, pp. 2196–2203.

[20] T. Tuytelaars and C. Schmid, “Vector quantizing feature space with a regular lattice,” in *Proc. IEEE Int. Conf. Comput. Vis.*, 2007, pp. 1–8.

[21] Z. Lei, Z. Yongdong, H. Richang, and Q. Tian, “Full-space local topology extraction for cross-modal retrieval,” *IEEE Trans. Image Process.*, vol. 24, no. 7, pp. 2212–2224, Jul. 2015.

[22] H. Jegou, M. Douze, and C. Schmid, “Product quantization for nearest neighbor search,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, pp. 117–128, Jan. 2011.

[23] J. L. Bentley, “Multidimensional binary search trees used for associative searching,” *Commun. ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[24] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 1984, pp. 47–57.

[25] N. Katayama and S. Satoh, "The SR-tree: An index structure for high-dimensional nearest neighbor queries," *ACM SIGMOD Rec.*, vol. 26, no. 2, pp. 369–380, 1997.

[26] P. Zezula *et al.*, *Similarity Search*. New York, NY, USA: Springer, 2006.

[27] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proc. 13th Annu. ACM Symp. Theory Comput.*, 1998, pp. 604–613.

[28] M. Datar *et al.*, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proc. 20th Annu. Symp. Comput. Geometry*, 2004, pp. 253–262.

[29] S. Kim and S. Choi, "Bilinear random projections for locality-sensitive binary codes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1338–1346.

[30] G. Ye, D. Liu, J. Wang, and S. Chang, "Large-scale video hashing via structure learning," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2013, pp. 2272–2279.

[31] M. Bawa, T. Condie, and P. Ganesan, "LSH forest: Self-tuning indexes for similarity search," in *Proc. Int. Conf. World Wide Web*, 2005, pp. 651–660.

[32] M. Norouzi, A. Punjani, and D. J. Fleet, "Fast exact search in hamming space with multi-index hashing," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 6, pp. 1107–1119, Jun. 2014.

[33] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental Bayesian approach tested on 101 object categories," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, vol. 106, no. 1, p. 178, 2004.

[34] L. I. Smith, "A tutorial on principal components analysis," *Inf. Fusion*, vol. 51, no. 3, pp. 219–226, 2002.

[35] J. Yang, D. Zhang, A. F. Frangi, and J.-Y. Yang, "Two-dimensional PCA: A new approach to appearance-based face representation and recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 1, pp. 131–137, Jan. 2004.

[36] M. Raginsky and S. Lazebnik, "Locality-sensitive binary codes from shift-invariant kernels," in *Proc. Adv. Neural Inf. Process. Syst.*, 2009, pp. 1509–1517.

[37] M. Datar *et al.*, "Locality sensitive hashing scheme based on p-stable distributions," in *Proc. 20th Annu. Symp. Comput. Geometry*, 2004, pp. 253–262.

[38] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 282, no. 3, pp. 1753–1760, 2008.

[39] Y. Weiss, R. Fergus, and A. Torralba, "Multidimensional spectral hashing," in *Proc. Eur. Conf. Comput. Vis.*, 2012, pp. 340–353.

[40] Y. Gong and S. Lazebnik, "Iterative quantization: A procrustean approach to learning binary codes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2011, pp. 817–824.

[41] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin, "Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 12, pp. 2916–2929, Dec. 2013.

[42] M. Muja and D. G. Lowe, "Fast matching of binary features," in *Proc. Comput. Robot Vis.*, 2012, pp. 404–410.

[43] C. C. Yan *et al.*, "Fast approximate matching of binary codes with distinctive bits," *Frontiers Comput. Sci.*, vol. 9, no. 5, pp. 741–750, 2015.

[44] G. Albert, P. Florent, G. Yunchao, and S. Lazebnik, "Asymmetric distances for binary embeddings," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 1, pp. 33–47, Jan. 2014.

[45] X. J. Wang, L. Zhang, F. Jing, and W. Y. Ma, "AnnoSearch: Image auto-annotation by search," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2006, pp. 1483–1490.

[46] X. Zhang, L. Zhang, and H. Y. Shum, "QsRank: Query-sensitive hash code ranking for efficient ε-neighbor search," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2012, pp. 2058–2065.

[47] W. Kong, W. J. Li, and M. Guo, "Manhattan hashing for large-scale image retrieval," in *Proc. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2012, pp. 45–54.

[48] W. Kong, "Double-bit quantization for hashing," in *Proc. AAAI Conf. Artif. Intell.*, 2013, pp. 137–138.

[49] S. Moran, V. Lavrenko, and M. Osborne, "Neighborhood preserving quantization for LSH," in *Proc. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2013, pp. 1009–1012.

[50] S. Moran, V. Lavrenko, and M. Osborne, "Variable bit quantization for LSH," in *Proc. 51st Annu. Meeting Assoc. Comput. Linguistics*, 2013, pp. 753–758.

[51] H. Zhu, "K-means based double-bit quantization for hashing," in *Proc. IEEE Symp. Comput. Intell. Multimedia, Signal Vis. Process.*, 2014, pp. 1–5.

[52] C. Xiong, W. Chen, G. Chen, D. Johnson, and J. J. Corso, "Adaptive quantization for hashing: An information-based approach to learning binary codes," in *Proc. SIAM Int. Conf. Data Mining*, 2014, pp. 172–180.

[53] Z. Wang *et al.*, "Hamming compatible quantization for hashing," in *Proc. Int. Joint Conf. Artif. Intell.*, 2015, pp. 2298–2304.

[54] X. Shen *et al.*, "Multi-view latent hashing for efficient multimedia search," in *Proc. ACM Int. Conf. Multimedia*, 2015, pp. 831–834.

[55] X. Shen *et al.*, "Semi-paired discrete hashing: Learning latent hash codes for semi-paired cross-view retrieval," *IEEE Trans. Cybern.*, vol. 47, no. 12, pp. 4275–4288, Dec. 2017.

Hongtao Xie received the Ph.D. degree in computer application technology from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2012. He is currently a Research Professor with the School of Information Science and Technology, University of Science and Technology of China, Hefei, China. His research interests include multimedia content analysis and retrieval, similarity search, and parallel computing.

Zhendong Mao received the Ph.D. degree in computer application technology from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2014. His research interests include the fields of multimedia, machine learning, and artificial intelligence.

Yongdong Zhang (M'08–SM'13) received the Ph.D. degree in electronic engineering from Tianjin University, Tianjin, China, in 2002. He is currently a Professor with the School of Information Science and Technology, University of Science and Technology of China, Hefei, China. He has authored more than 100 refereed journal and conference papers. His research interests include multimedia content analysis and understanding, multimedia content security, video encoding, and streaming media technology.

Prof. Zhang was the recipient of the Best Paper Awards in PCM 2013, ICIMCS 2013, and ICME 2010, and the Best Paper Candidate in ICME 2011. He serves as an Editorial Board Member of the *Multimedia Systems Journal* and the IEEE TRANSACTIONS ON MULTIMEDIA.

Han Deng received the M.A.Eng. degree from the Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China, in 2017. She is currently with the ByteDance AI Lab., Beijing, China. Her research interests include multimedia content retrieval, similarity search, and semantic segmentation.

Chenggang Yan received the B.S. degree from Shandong University, Jinan, China, in 2008, and the Ph.D. degree from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2013, both in computer science.

He is currently a Professor with Hangzhou Dianzi University, Hangzhou, China. Before that, he was an Assistant Research Fellow with Tsinghua University. His research interests include machine learning, image processing, computational biology, and computational photography. He has authored or co-authored more than 30 refereed journal and conference papers. As a co-author, Prof. Yan was the recipient of the Best Paper Awards in the International Conference on Game Theory for Networks 2014, and SPIE/COS Photonics Asia Conference 9273 2014, and the Best Paper Candidate in the International Conference on Multimedia and Expo 2011.

Zhineng Chen received the B.S. and M.S. degrees from the College of Information Engineering, Xiangtan University, Xiangtan, China, in 2004 and 2007, respectively, and the Ph.D. degree from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2011, all in computer science. He is currently an Associate Professor with the Institute of Automation, Chinese Academy of Sciences. He was a Senior Research Associate with the Department of Computer Science, City University of Hong Kong, in 2012, and was an Assistant Professor with the Institute of Automation, Chinese Academy of Sciences, from 2012 to 2014. His research interests include large-scale multimedia information retrieval, machine vision, and pattern recognition. He has authored or co-authored more than 20 papers in prestigious multimedia and related conferences.