Fast Search with Data-Oriented Multi-Index Hashing for Multimedia Data

Yanping Ma[1,2], Hailin Zou[1], Hongtao Xie[3], Qingtang Su[1]

[1]Scholol of Information and Electrical Engineering, Ludong University, Yantai, China
[2]Key Laboratory of Intelligent Information Processing, Chinese Academy of Sciences (CAS), Beijing, China
[3]Institute of Information Engineering, CAS, National Engineering Laboratory for Information Security Technologies, Beijing, China
[e-mail: myp74920@126.com, zhyd@ict.ac.cn, xiehongtao@iie.ac.cn]
*Corresponding author: Yanping Ma

Received January 3, 2015; revised May 8, 2015; accepted June 2, 2015; published July 31, 2015

Abstract

Multi-index hashing (MIH) is the state-of-the-art method for indexing binary codes, as it divides long codes into substrings and builds multiple hash tables. However, MIH is based on the dataset codes uniform distribution assumption, and will lose efficiency in dealing with non-uniformly distributed codes. Besides, there are lots of results sharing the same Hamming distance to a query, which makes the distance measure ambiguous. In this paper, we propose a data-oriented multi-index hashing method (DOMIH). We first compute the covariance matrix of bits and learn adaptive projection vector for each binary substring. Instead of using substrings as direct indices into hash tables, we project them with corresponding projection vectors to generate new indices. With adaptive projection, the indices in each hash table are near uniformly distributed. Then with covariance matrix, we propose a ranking method for the binary codes. By assigning different bit-level weights to different bits, the returned binary codes are ranked at a finer-grained binary code level. Experiments conducted on reference large scale datasets show that compared to MIH the time performance of DOMIH can be improved by 36.9%–87.4%, and the search accuracy can be improved by 22.2%. To pinpoint the potential of DOMIH, we further use near-duplicate image retrieval as examples to show the applications and the good performance of our method.

Keywords: Nearest Neighbor Search, Binary Codes, Indexing

This work is supported by the "Strategic Priority Research Program" of the Chinese Academy of Sciences (XDA06030602), National High Technology Research and Development Program (2011AA010705), National Nature Science Foundation of China (61170161, 61303171, 61271406), Natural Science Foundation of Shandong Province (ZR2014FM005). We would like to thank Dr. Mohammad Norouzi (University of Toronto) for providing the code of [7] and for giving precious suggestions.

http://dx.doi.org/10.3837/tiis.2015.07.015 **ISSN : 1976-7277**

1. Introduction

Nearest neighbor (NN) search in large-scale dataset is a fundamental requirement in many applications, including image retrieval [1], object recognition [2] and computer vision [3]. Recently, many works represent visual content and feature descriptors in terms of compact binary codes [4-6], as they are storage efficient and comparisons require only a few machine instructions. With binary codes, a query in a dataset of millions of items can be accomplished in less than a second [7].

Although the distances between binary codes in the Hamming space can be calculated efficiently, linear search will lose its performance in front of large-scale dataset. This is because that the computing power of the processor is limited, while the size of dataset can be very huge. To improve the performance of NN search with binary codes, binary hashing has been proposed [4, 8] and is becoming increasingly popular. The binary hashing uses binary codes as direct indices (memory addresses) into a hash table. Then, the nearest neighbors can be found by exploring a set of hash table buckets within some Hamming ball around the query. With progressively increasing the search radius, exact nearest neighbors of the query can be found [8]. However binary hashing has a serious problem that is the number of hash buckets grows near-exponentially with the search radius. So it is efficient only when the length of codes is less than 32 bits [7]. When the codes are long and even with a small search radius, the number of buckets to examine will larger than the size of database. So it result in slower than linear scan.

To improve the ability of binary hashing in dealing with long codes, Norouzi et al. [7] propose multi-index hashing (MIH) method. MIH divides the long codes into several disjoint but consecutive binary substrings and build multiple hash tables, one table for each substring. Given a query code, it is also divided into query substrings, and candidate neighbors are found by using query substrings as direct indices into their corresponding hash tables. Finally, candidates are checked for validity using the entire binary code, to remove any non nearest neighbors. For each table (substrings), the dimension of Hamming space and radius of the searching Hamming ball are only a fraction of that for the long code. Thus, MIH enormously reduces the number of buckets to be checked and improves the search efficiency.

While favorable for simplicity and scalability, MIH has two shortcomings. First, it is based on the assumption that the codes in the dataset are distributed uniformly [7]. Actually, the codes are not uniformly distributed, especially for the multimedia data [1]. So, the time performance of MIH will be adversely affected when dealing with non-uniformly distributed datasets [7,11]. Second, as Hamming distance is discrete and bounded by the code length, in practice, there are a lot of data items sharing the same Hamming distance to the query and the ranking of these data items is ambiguous [13]. So it poses a critical issue for similarity search, where ranking is important. But, most existing binary hashing methods lack in providing a good ranking of results [14].

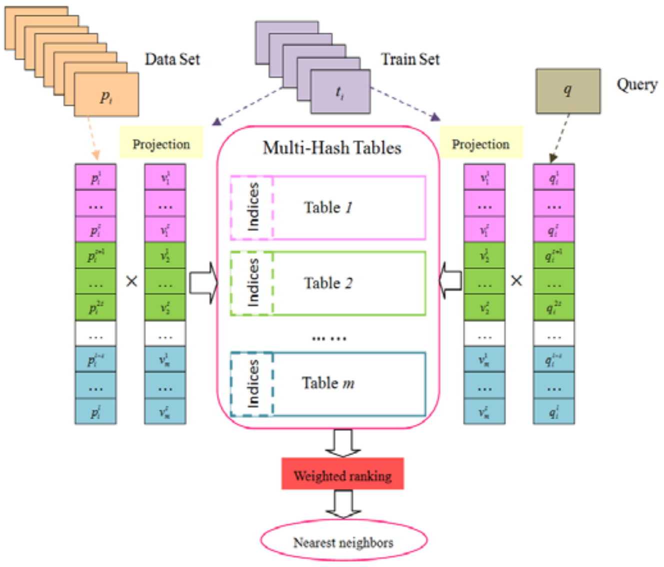


Fig. 1. The framework of our method. We divide the long codes into several disjoint but consecutive substrings. But we project the substrings with learned projection vectors to generate indices in hash tables. The length of the codes is *l*, the number of substrings is *m*, and each substring has *s* bits. The vector *v* is the learned projection vector. For candidate results, we also propose a ranking method. By assigning different bit-level weights to different bits, the results are ranked to improve the search accuracy.

In this paper, we propose a data-oriented multi-index hashing (DOMIH) method to solve the aforementioned two problems. The framework of our method is illustrated in **Fig. 1**. We also divide the long codes into several disjoint but consecutive substrings in building multiple hash tables. But our method has three major differences between MIH. Firstly, we build a training set to compute the correlations (covariance matrix) between bits of the codes and learn an adaptive projection vector (vector *v*) for each substring. Then, instead of using binary substrings as direct indices into a hash table, we project the substrings with corresponding projection vectors to generate new indices. With adaptive projection, the data items in each hash table of DOMIH are more near uniformly distributed than that in MIH. So it handles the "non-uniform distribution" problem to some extent. Finally, we propose a ranking method for the binary codes with the covariance matrix. By assigning different bit-level weights to different bits, the returned binary codes are ranked at a finer-grained binary code level.

The training process is accomplished offline, and the calculation of projection for binary codes can be ignored, so DOMIH brings negligible computational overhead compared to MIH. As DOMIH generates more uniform codes distribution in each hash table than MIH and refines the ranking of results, it can obviously improve the search efficiency and accuracy. Experiments conducted on reference datasets show that compared to MIH the time performance of DOMIH can be improved by 36.9%–87.4%, and the search accuracy can be improved by 22.2%.

The rest of this paper is organized as follows. Section 2 gives a review of previous works. Section 3 presents the proposed DOMIH method. Section 4 shows the experimental comparisons. In section 5, we give the conclusion and future work.

2. Literature Review

Recently, many works represent visual content and feature descriptors in terms of compact binary codes [4-6], as they are storage efficient and comparisons require only a few machine instructions. Accordingly, many index structures designed for binary codes are proposed, and they can be classified into approximate nearest neighbor (ANN) search and exact search.

For ANN search with binary codes, many algorithms [12, 15, 16, 17]designed for floating-point vectors are not suitable [18]. Rublee *et al*. [6] choose locality sensitive hashing (LSH) as nearest neighbor search for binary codes. In LSH, points are stored in several hash tables and hashed in different buckets. Given a query descriptor, its matching buckets are retrieved and its elements compared using a brute force matching. For binary codes, the hash function is simply a subset of the signature bits: the buckets in the hash tables contain descriptors with a common sub-signature. To assure high search accuracy, the method in [6] has to build many hash tables and it costs too much memory. So it has limited scalability. Zitnick uses min-hash [21] as an efficient mean for finding similar neighbors [22]. Min-hash has the property that the probability of two hashes being identical is equal to the Jaccard similarity, and binary codes are good candidate for the min-hash algorithm. However, Min-hash has limited scalability as LSH. Esmaeili *et al.* propose Error Weighted Hashing (EWH) algorithm [23] for the Hamming space. EWH significantly reduces the number of candidate nearest neighbors by weighing them based on the difference between their hash vectors. Zhang *et al*. present an efficient query processing method for Hamming distance queries, which is called HmSearch [24]. HmSearch is based on improved enumeration-based signatures, enhanced filtering, and the hierarchical binary filtering-and-verification. Muja *et al*. [18] introduce a new algorithm for approximate matching of binary codes, based on priority search of multiple hierarchical clustering trees. However, as they randomly select cluster centers and use the entire codes for index building and searching, the efficiency is limited.

Linear search [1] is an intuitive and typical exact search method. Given a query, the brute-force matching is performed to find the nearest neighbors. In the Hamming space, a query in a dataset of millions of items can be accomplished in less than a second [7]. However, using linear search for matching becomes a bottleneck for large datasets. To improve the efficiency of nearest neighbor search with binary codes, binary hashing has been proposed [8]. It uses binary codes as direct indices (addresses) into a hash table. Then, the nearest neighbors can be found by exploring a set of buckets within the Hamming ball around the query. With gradually increasing the search radius, exact nearest neighbors can be found [8]. But it has a fatal problem that the number of buckets grows near-exponentially with the search radius. So it is efficient only when the length of codes is less than 32 bits [7]. When the codes are long and even with a small search radius, the number of buckets to explore will larger than the size of database. So it result in slower than linear search. To improve the ability of binary hashing in dealing with long codes, Norouzi *et al.* [7] propose MIH method, which divides the long codes into several disjoint but consecutive binary substrings and build multiple hash tables. Given a query, the candidate neighbors are found by searching query substrings in corresponding hash tables. MIH is based on the assumption that the codes in the dataset are distributed uniformly [7]. So, the time performance of MIH will be adversely affected when dealing with non-uniformly distributed datasets [11].

In most existing binary indexing methods, including the works discussed above, the returned results are simply ranked based on the Hamming distance to the query. As Hamming distance metric gives each bit the same weight, it unable to distinguish between the relative importance of different bits and causes ambiguity for ranking [13]. One way to avoid this ambiguity is assigning different bit-level weights to different bits. The weighted Hamming distance has been used for image retrieval, such as AnnoSearch [14] and Hamming distance weighting [19]. In AnnoSearch, each bit of the binary code is assigned with a bit-level weight; while in Hamming distance weighting, the aim is to weight the overall Hamming distance of local features for feature matching. In these works [14,19], only a single set of weights is used to measure either the importance of each bit in Hamming space, or to rescale the Hamming distance for better image matching. In [20], the authors propose a query-sensitive ranking algorithm QsRank. For a query, QsRank assigns two weights to each bit and defines a score function to measure the confidence of the neighbors of a query mapped to a binary code. The returned codes are ranked based on their scores. Zhang *et al.* [13] propose a bit-level weighting method WhRank, which is not only data-dependent, but also query-dependent. Experimental comparisons demonstrate the performance of WhRank, but the query-dependent computation affects the query efficiency and the data-dependent procedure is based on the hash functions, which is not applicable for pure binary codes.

3. DATA-ORIENTED MULTI-INDEX HASHING

In this section, we first briefly explain the MIH [7] method. Then, the proposed DOMIH will be elaborated in detail. We also put forward an entropy based measurement to evaluate the distribution of data items in each hash table. Finally, the ranking method for binary codes is presented.

3.1 Multi-Index Hashing

Although the matching between binary codes in the Hamming space can be calculated with only a few XOR operations, which can be efficiently executed by the processor, linear search cannot handle large-scale dataset. Thus, many literatures [4, 7, 8] propose binary hashing method, which incorporates binary codes and hash tables[9].

Binary hashing converts the NN search problem in Hamming space to the *R*-near neighbor problem:

$$\|q - p_i\|_H \le r,\ i = 1, 2, \dots, N,\ p_i \in D\ , \tag{1}$$

where q is the query item, D is the dataset, N is the size of D and $\|\bullet\|_H$ denotes the Hamming distance. With progressively increasing r, it can find the *R*-near neighbor of q. To accelerate the search speed, binary hashing builds a hash table and uses the binary codes as the direct indices (memory addresses) of the buckets. Then, the *R*-near neighbor problem can be solved by exploring a Hamming ball centered at q with radius r. Thus the number of buckets to be checked is:

$$num = \sum_{i=0}^{r} C_l^i\ , \tag{2}$$

where l is the dimension of the binary codes. When the codes are long and even with a moderate value of r, the number of buckets to check will increase exponentially [7]. **Sometimes** this number will larger than the size of database. For example when l=256, r=5, the number of checked buckets is $num \approx 10^{10}$. In practice, the length of binary codes is often longer than or equal to 256 (such as ORB [6]), and the search radius is usually larger than 10 [7], in order to achieve satisfactory retrieval performance. In this case, binary hashing is slower than linear scan.

To handle the above problem, MIH method [7] is proposed. The idea of MIH is very simple and clear. It divides the long code of l bits into m disjoint but consecutive binary substrings, and the length of substrings is equal to $\lceil l/m \rceil$ or $\lfloor l/m \rfloor$. Then, MIH builds m hash tables, one table for each substring. In query, the query code is also divided into substrings, and candidate neighbors are found by using query substrings as indices in corresponding hash tables. Finally, candidates are checked for validity using the entire binary code. The theoretical basis of MIH is: when two binary codes q and p differ by r bits or less, then in at least one of their m substrings they must differ by at most $\lfloor r/m \rfloor$ bits:

$$\exists\, 1 \le k \le m \;\; s.t. \;\; \left\| q^k - p^k \right\|_H \le \lfloor r/m \rfloor , \tag{3}$$

where q^k is the *k-th* substring of q. In this way, MIH can significantly reduce the number of checked buckets compared to binary hashing. For example, if l=256, r=5 and m=2, then the total number of buckets to be checked is $m * \sum_{i=0}^{2} C_{128}^{i}$ =16,512, which is much less than 10^{10}. So MIH has obvious advantage in indexing long codes.

However, MIH is based on the assumption that the codes in the dataset are uniformly distributed [7]. Actually, the codes are not uniformly distributed, especially for the multimedia data [1, 11]. Besides, there are a lot of data items sharing the same Hamming distance to a query and the ranking of these data items is ambiguous. So MIH has these shortcomings:

- For the candidate buckets in the multi-hash tables, if they have too many items, then there are too many candidate codes need checking for validity. So it costs much time for candidate codes checking.
- For the candidate buckets in the multi-hash tables, if they have too few items, then in search process the value of r needs set to be large enough to ensure that enough exact near neighbors are found. So it costs much time for index lookup.
- For the applications, such as image retrieval and computer vision, where ranking of data items is important, MIH cannot distinguish the binary codes sharing the same Hamming distance to the query.

For these reasons, the efficiency of MIH depends on the distribution of database codes. To reach the optimal time performance, we should make the database items are evenly assigned to hash buckets. Besides, we also need to measure the distinctiveness of different bits for result ranking.

3.2 Data-Oriented Multi-Index Hashing

In MIH, the binary substrings are used as direct indices into multi-hash tables. As the database codes are not uniformly distributed, the substrings are also not uniformly distributed. Thus the data items in multi-hash tables are distributed unevenly. To deal with this problem, we propose DOMIH method, which is built on MIH but consists of two additional steps, as illustrated in **Fig. 1**. Firstly, a training set is built to compute the correlations between bits of the codes and learn an adaptive projection vector for each substring. Then, instead of using binary substrings as direct indices, we project the substrings with corresponding projection vectors to generate indices. With adaptive projection, the indices in each hash table of DOMIH are more approximately uniformly distributed than that in MIH.

The non-uniform distribution of database codes is caused by the correlations between code bits [7, 11]. Principal component analysis (PCA) [10] is a statistical method that uses orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. The first principal component has the largest possible variance, which accounts for as much of the variability in the data as possible. Therefore, we use the first principal component of PCA for code bits decorrelation. The steps are as follows:

1.Build a training set $T = [x_1, x_i, \cdots, x_n]$ for PCA learning, and x_i is a *l*-dimensional binary column vector, which denotes a binary code. The training set has no elements in common with the base set.

2.Compute mean value of T and the covariance matrix S :

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \ , S = \sum_{i,j=1}^{n} (x_i - \mu)(x_j - \mu)^T \ . \tag{4}$$

3.For each substring, we first get its covariance matrix $S^{'}$, which is the submatrix of S .Then we execute the eigenvalue decomposition of matrix $S^{'}$ and take the eigenvector V , which is corresponding to the largest eigenvalue of $S^{'}$, as the adaptive projection vector for this substring.

4.In index building and searching, instead of using binary substrings as direct indices, we project the substrings with corresponding projection vectors to generate new indices. Suppose the substring of the code is $p = [p_1, p_i, ..., p_d]$, the indice is calculated as:

$$indice = \sum_{i=1}^{d} p_i * v_i * 2^{d-i} \ , \tag{5}$$

where *v* is the projection vector.

As projection vectors contain data distribution information and are data-oriented, the multi-hash tables are also data-oriented. By using PCA to align the indices, we can assume an approximate uniform distribution of database codes.

3.3 Distribution Evaluation

To evaluate the distribution of data items in each hash table, we utilize an entropy based measurement. For a hash table *h*, suppose that the size of database is *N*, there are *num_b* buckets in this table and the number of codes in bucket i is $n(i)$. The probability of codes assigned to bucket *i* is estimated as $p(i) = n(i)/N$. Then the entropy of hash table *h* is defined as:

$$E(h) = -\sum_{i=1}^{num\_b} (p(i) * \log(p(i))) \tag{6}$$

Intuitively, higher $E(h)$ indicates better distribution of data items in hash table. With this measurement, we can quantize and compare the distribution of hash tables.

3.4 Bit-level Weighted Ranking

In this sub-section, we present the bit-level weighted ranking algorithm. In most binary indexing algorithms, the distance between two data items is simply measured by the Hamming distance between their binary codes. This distance metric is somewhat ambiguous, since for a *l*-bits binary code *p*, there are C_l^d different binary codes sharing the same distance *d* to *p*. For the applications, such as image retrieval and computer vision, where ranking of data items is important, it should rank the matching features sharing the same distance. But in most binary indexing algorithms, each bit takes the same weight and makes the same contribution to distance calculation.

With the covariance matrix S , we propose a simple but effective bit-level weighted ranking algorithm. Suppose the elements in the main diagonal of S are $(\lambda_1, \lambda_2, ..., \lambda_l)$, and l is the length of the binary code, then the weight for the *i-th* bit is defined as:

$$w_i = \frac{\lambda_i}{\sum_{j=1}^{l} \lambda_j} \tag{7}$$

We use the discriminating power of a bit to measure its bit-level weight. For example, for a query q and two data items p_1 , p_2 sharing the same Hamming distance d (=1) to q, where p_1 and p_2 are different with q on bits k_1 and k_2 respectively. If bit k_1 is more discriminative than k_2 , then p_2 is considered to be more similar with q than p_1 , as the bit-flipping on bit k_1 gives a higher confidence that p_1 is not a neighbor of q than that on k_2 . Thus, w_{k_1} should be larger than w_{k_2} , which means the more discriminative a bit k is, the larger the associated weight w_k is. Essentially, the larger the value of λ_i , the more discriminative of the *i-th* bit [10]. So we just use this property to distinguish different bits. This bit-level weighted ranking algorithm is intuitive, but it is effective, as illustrated by the experimental comparison.

4. EXPERIMENTS

In this section, experiments are conducted to evaluate the proposed DOMIH method. First, we investigate the influence of the size of training set, and compare the overall search performance with the state-of-the-art methods on famous benchmark datasets. Then, the bit-level weighted ranking algorithm is evaluated. Finally we use near-duplicate image retrieval as examples to show the applications of DOMIH and demonstrate the good performance of our method.

4.1 Dataset and Evaluation Protocol

The experiments are conducted on the famous ANN_SIFT1B dataset and the item is 128-D SIFT descriptor [12]. It contains a training set of 10^8 descriptors, a query set of 10^4 descriptors and a base set of 10^9 descriptors. We use the source code, minimal loss hashing, provided by Norouzi *et al.* [7] to create datasets of binary codes and each code has 128 bits. In experiments, the training set is used to learn the adaptive projection vectors. After learning is finished, the training data is removed and the base set and query set is used for experimental comparison. So the training set has no elements in common with the base set.

For the comprehensive evaluation of our method, we compare the performances of the following approaches: 1) MIH [7], the standard multi-index hashing method; 2) DDMIH [11], data driven multi-index hashing method by separating the correlated bits into different inconsecutive segments; 3) DOMIH, our method proposed in this paper. The experimental environment is Intel Xeon E5-2620*2(2.00 GHz, 7.2GT/s, 15M cache, 6cores) and with 64 GB memory. All the three methods have the same experimental setup, such as the hardware environment and the number of substrings.

Table 1. Recall @ 1000 near neighbors of DOMIH with different sizes of training set. The evaluation is conducted on the database of 10^9 codes.

size of training set	10^3	10^4	10^5	10^6	10^7	10^8
$recall@1000NN$	0.447	0.524	0.572	0.623	0.626	0.627

4.2 Influence of the Size of Training Set

When we build training set to learn the adaptive projection vectors, intuitively the larger size of training set the more accurate projection vectors can be obtained. However, lager training set will result in high computational overhead, even the learning process is executed offline. **Table 1** shows the recall @ 1000 near neighbors of DOMIH with different sizes of training set. It is defined as follows:

$$recall@1000NN = \frac{DOMIH\_1000NN \cap Linear\_1000NN}{Linear\_1000NN}, \quad (8)$$

where $DOMIH\_1000NN$ is the raw 1000 near neighbors returned by DOMIH, without sorting with the full Hamming distance, and $Linear\_1000NN$ is the 1000 near neighbors returned by linear scan. For each query, we calculate its $recall@1000NN$, and then take the mean value over all queries.

We enumerate the size of training set from 10^3 to 10^8, and find that the performance is improved with larger size. However, the improvement is not significant when the size is too large. When the size of training set is 10^6, the performance becomes stable. As computing covariance matrix costs much time, we select the 10^6 training set for efficiency concern.

Table 2. Distribution of data items in the hash tables of MIH, DDMIH and DOMIH, which is quantized by entropy based measurement. The evaluation is conducted on the dataset of 10^9 items.

	MIH	DDMIH	DOMIH
Table 1	4.372	4.851	6.243
Table 2	4.201	4.526	6.188
Table 3	4.178	4.535	6.215
Table 4	4.353	4.627	6.287

4.3 Comparison of Search Efficiency

To evaluate the efficiency of DOMIH, we first compare it to MIH and DDMIH in distribution of data items in hash tables, which is quantized by equation (6). Then, the search radii that are required to find 1000-NN on 1B 128-bit binary codes for these three methods are compared. Finally, the time costs with respect to database size N and the number of near neighbors R are illustrated. As there are 10^9 codes in the database, the length of the substring is $\log_2^{10^9} \approx 32$ [7]. So there are $128/32=4$ hash tables. **Table 2** shows the $E(h)$ value for each hash table of these methods. From **Table 2**, we can see that all the hash tables in DOMIH have larger entropy value than the other two methods. So the indices in each hash table of DOMIH are more approximatively uniformly distributed than that in MIH and DDMIH.

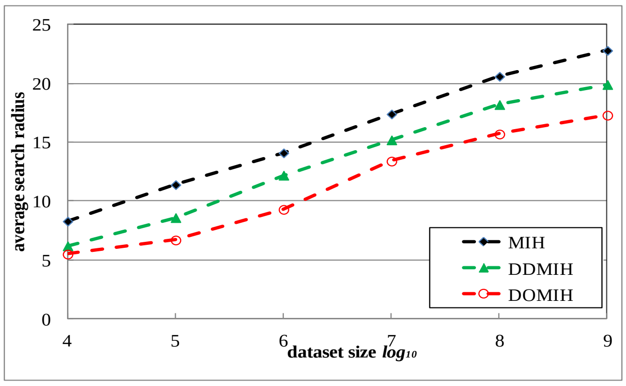


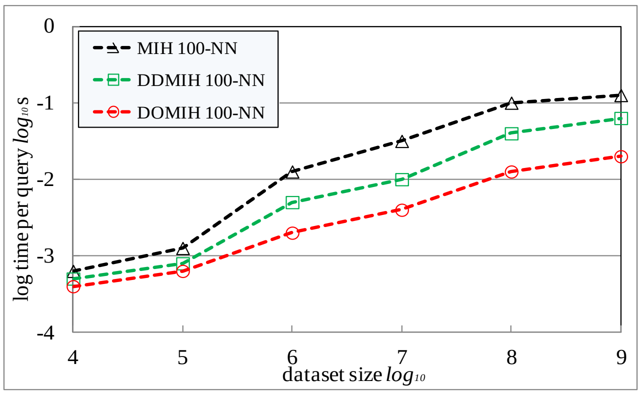
Fig. 2. Average search radii required to find 1000-NN on 1B 128-bit binary codes for DOMIH, MIH and DDMIH.

To use the multi-index hashing in practice, one has to specify a Hamming search radius *r*. For many applications, the value of *r* is chosen such that queries will retrieve *k* near neighbors on average. Nevertheless, a fixed radius for all queries would produce too many neighbors for some queries, and too few for others. It is therefore more natural for many tasks to fix the number of required neighbors, *i.e.*, *k*, and let the search radius depend on the query. Given a query, one can progressively increase the Hamming search radius per substring, until a specified number of neighbors are found. Intuitively, smaller search radius means faster search speed, as it has less index query operation.

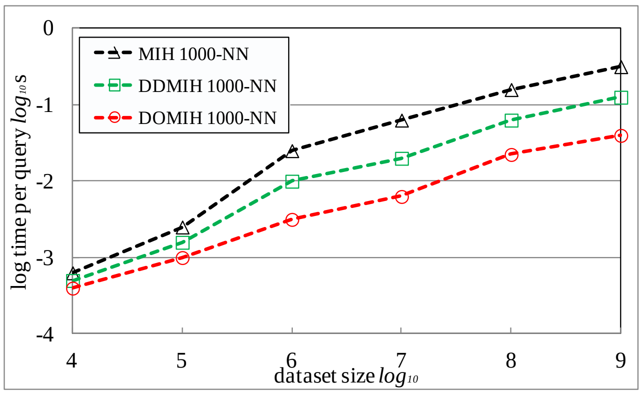
Fig. 2 depicts the average search radii required to find 1000-NN on 1B 128-bit binary codes for DOMIH, MIH and DDMIH. From figure 2, we can see that our DOMIH has smaller search radii than MIH and DDMIH in all cases. This indicates the better search efficiency of DOMIH.

Fig. 3 shows the time costs of these methods with respect to different data size and different number of returned near neighbors. From Fig. 3, we get three major observations:

- It is clear that the proposed DOMIH achieves the best efficiency. Compared to MIH, the time performance of DOMIH can be improved by 36.9%–87.4% on the 1B dataset.
- Besides, DOMIH is always faster than DDMIH. This is because that our method can remove the correlations between bits much better.
- From Fig. 3, we can also observe that, with the increasing of dataset (10^4-10^9) the time cost of DOMIH increases slower than the other two methods, especially for the large-scale dataset. This reflects the good time performance of our method.



(a)



(b)

Fig. 3. Run-times per query for MIH, DDMIH and DOMIH with 100 (a) and 1000 (b) nearest neighbors on 1B 128-bit binary codes.

4.4 Evaluation of Bit-level Weighted Ranking

In this sub-section, we evaluate the proposed bit-level weighted ranking algorithm. As MIH solves exact nearest neighbor search in Hamming distance, we use Euclidean NN recall rate for *k*-NN search on binary mappings of 1B SIFT descriptors [7] as performance metric, which is defined as：

$$Eucli.\ NN\ recall@k = \frac{Hamm.NN}{Eucli.NN}, \tag{9}$$

where $Eucli.NN$ is the result of the nearest neighbor search of SIFT descriptors in Euclidean distance, and $Hamm.NN$ is the result of the nearest neighbor search of corresponding binary codes in Hamming distance. Note that the length of the binary code is 128-bit. **Fig. 4** shows the $Eucli.\ NN\ recall@k$ of our DOMIH and MIH. We can clearly see that with bit-level weighted ranking, DOMIH always has higher recall rate than MIH. So our method is useful for the applications, such as image retrieval and computer vision, where ranking of matching features is important.

Averagely, compared to MIH the $Eucli.\ NN\ recall@k$ of DOMIH can be improved by 22.2% on the 1B dataset.

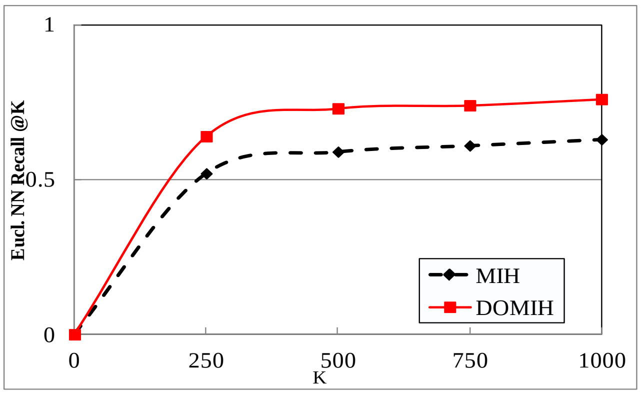


Fig. 4. Recall rates for1B dataset obtained by k-NN on 128-bit binary codes.

4.5 Near-duplicate Image Retrieval

In this experiment we show an application of DOMIH in near-duplicate image retrieval, which plays an important role in many applications. The experiment is conducted on Ukbench dataset [25], Holiday dataset [19] and Oxford dataset [26]. There are 10200, 1491 and 5062 images in these sets, respectively. The images in these sets are modified through a serial of photometric and geometric transformations, such as blurring, cropping, adding noise, the changes of lighting, viewpoint, color and camera lens. They are the benchmark datasets for near-duplicate image retrieval evaluation. We also download 50 thousands images from Flickr[1]as distracting images.

The near-duplicate image retrieval system is built upon the scheme proposed by [6]. The local features are obtained by ORB [6], and the feature sets of the database images are indexed by MIH [7], DDMIH [11] and DOMIH, respectively. Note that, we just compare the performance of different index structures. So the post-processing methods, such as geometric consistency constraint and spatial coherent verification [27, 28], are not applied. The performance metric applied in our experiment is mean average precision (mAP) [19]. We compare the following three approaches: 1) ImR-MIH, retrieval system with the binary index structure proposed in [7]; 2) ImR-DDMIH [11], retrieval system with the binary index structure proposed in [11]; and 3) ImR-DOMIH, retrieval system with the binary index structure proposed in this paper.

Table 3. Retrieval accuracy for three index structures on Ukbench [25], Oxford [26], and Holiday [19] datasets. The performance metric applied is mean average precision.

	Ukbench	Oxford	Holiday
ImR-MIH	0.702	0.483	0.677
ImR-DDMIH	0.748	0.534	0.743
ImR-DOMIH	0.793	0.587	0.784

Table 3 compares the retrieval accuracy of the above three systems. We can see that DOMIH significantly improves the search precision. Compared to ImR-DDMIH, our method achieves 6%, 10% and 5.5% improvements in mAP on the three sets respectively. Compared to the ImR-MIH, the search accuracy gain obtained by our method is obvious.

Table 4. Search efficiency for three index structures on Ukbench [25], Oxford [26], and day [19] datasets. The experimental environment is Intel Xeon E5-2620*2(2.00 GHz, 7.2GT/s, 15M cache, 6cores) and with 64 GB memory.

	Ukbench	Oxford	Holiday
ImR-MIH	21.3s	19.1s	16.4s
ImR-DDMIH	18.7s	16.5s	15.2s
ImR-DOMIH	15.2s	13.4s	12.8s

[1] http://www.flickr.com/

Table 4 compares the retrieval efficiency of the above three systems. For efficiency concern, we resize the images and each image has about 500 features. For each query feature, we get its 100 nearest neighbors. We can see that DOMIH significantly improves the search speed. Compared to ImR-MIH, our method achieves 28.6%, 29.8% and 22% improvements in search efficiency on the three sets respectively.

5. CONCLUSION

We have introduced a data-oriented multi-index hashing method to solve the problems of the state-of-the-art methods: efficiency losing in handling non-uniformly distributed codes and without ranking for accurate search. By taking advantage of the statistics of the dataset, we first learn adaptive projection vector for each binary substring and project substrings with corresponding projection vectors to generate new indices. With adaptive projection, the indices in each hash table are near uniformly distributed. Then, by assigning different bit-level weights to different bits, the returned binary codes are ranked at a finer-grained binary code level. Experiments conducted on famous datasets show the obvious performance improvement of our method.

References

[1] Wei Zhang, Ke Gao, Yongdong Zhang and Jintao Li, "Efficient Approximate Nearest Neighbor Search with Integrated Binary Codes," in *Proc. of ACM International Conference on Multimedia*, pp. 1189-1192, 2011. Article (CrossRef Link)

[2] A. Torralba, R. Fergus and Y. Weiss, "Small codes and large image databases for recognition," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, 2008. Article (CrossRef Link)

[3] Lei Zhang, Yongdong Zhang, Jinhui Tang, Xiaoguang Gu, Jintao Li and Qi Tian, "Topology Preserving Hashing for Similarity Search," in *Proc. of ACM International Conference on Multimedia*, 2013. Article (CrossRef Link)

[4] R. Salakhutdinov and G. Hinton, "Semantic Hashing," *International Journal of Approximate Reasoning*, 2009. Article (CrossRef Link)

[5] C. Strecha, A. Bronstein, M. Bronstein and P. Fua, "LDAHash: improved matching with smaller descriptors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 1, pp. 66-78, 2012. Article (CrossRef Link)

[6] E. Rublee, V. Rabaud, K. Konolige and G. Bradski, "ORB: an efficient alternative to SIFT or SURF," in *Proc. of IEEE International Conference on Computer Vision*, pp. 2564-2571, 2011. Article (CrossRef Link)

[7] M. Norouzi, A. Punjani, and D.J. Fleet, "Fast search in hamming space with multi-index hashing," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, 2012. Article (CrossRef Link)

[8] Y. Weiss, A. Torralba and R. Fergus, "Spectral Hashing," *Advances in Neural Information Processing Systems*, 2008.

[9] Datar, M., Immorlica, N., Indyk, P.,*et al.*, "Locality-sensitive hashing scheme based on p-stable distributions," *SCG '2004*, 2004. Article (CrossRef Link)

[10] Jolliffe, I.T, "Principal Component Analysis," *second edition Springer*, 2002. Article (CrossRef Link)

[11] Ji Wan, Sheng Tang, Yongdong Zhang, Lei Huang and Jintao Li, "Data Driven Multi-Index Hashing," in *Proc. of IEEE International Conference on Image Processing*, 2013. Article (CrossRef Link)

[12] Jegou H., Douze M., et al. , “Product Quantization for Nearest Neighbor Search,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117-128, 2011 Article (CrossRef Link)

[13] Lei Zhang, Yongdong Zhang, Jinhui Tang, Ke Lu and Qi Tian, “Binary Code Ranking with Weighted Hamming Distance,” in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, 2013. Article (CrossRef Link)

[14] X.Wang, L. Zhang, F. Jing, and W. Ma, “AnnoSearch: Image auto-annotation by search,” in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, 2006. Article (CrossRef Link)

[15] M. Aly, M. Munich and P. Perona, “Distributed kd-trees for retrieval from very large image collections, In *Proc. of British Machine Vision Conference*, 2011.

[16] A. Babenko and V. Lempitsky, “The inverted multi-index,” In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, 2012. Article (CrossRef Link)

[17] C. SilpaAnan and R. Hartley, “Optimized KD-trees for fast image descriptor matching,” *CVPR*, 2008. Article (CrossRef Link)

[18] M. Muja and D. G. Lowe, “Fast matching of binary features,” *Computer and Robot Vision*, 2012. Article (CrossRef Link)

[19] H. Jegou, M. Douze, and C. Schmid, “Improving bag-of features for large scale image search,” *International Journal of Computer Vision*, no.87, pp. 316-336, 2010. Article (CrossRef Link)

[20] L. Zhang, X. Zhang, and H.-Y. Shum, “QsRank: Query sensitive hash code ranking for efficient neighbor search,” in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, 2012. Article (CrossRef Link)

[21] Broder, A.Z, “On the resemblance and containment of documents,” *IEEE Compression and Complexity of Sequences*, pp. 21-29, 1997. Article (CrossRef Link)

[22] C. Zitnick, “Binary coherent edge descriptors,” in *Proc. of European Conference on Computer Vision*, pp. 1-14, 2010. Article (CrossRef Link)

[23] Mani Malek Esmaeili, Rabab Kreidieh Ward and Mehrdad Fatourechi, “A Fast Approximate Nearest Neighbor Search Algorithm in the Hamming Space,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 34, no. 12, 2012. Article (CrossRef Link)

[24] Xiaoyang Zhang, Jianbin Qin, Wei Wang, Yifang Sun and Jiaheng Lu, “HmSearch: An Efficient Hamming Distance Query Processing Algorithm,” in *Proc. of International Conference on Scientific and Statistical DB Management*, 2013. Article (CrossRef Link)

[25] D. Nister and H. Stewenius, “Scalable recognition with a vocabulary tree,” in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, 2006. Article (CrossRef Link)

[26] J. Philbin, O. Chum, M. Isard, et al, "Object retrieval with large vocabularies and fast spatial matching,” in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, pp.1-8, 2007. Article (CrossRef Link)

[27] Hongtao Xie, Ke Gao,Yongdong Zhang, Sheng Tang, Jintao Li, Yizhi Liu, “Efficient Feature Detection and Effective Post-Verification for Large Scale Near-Duplicate Image Search,” *IEEE Trans. on Multimedia*, 2011. Article (CrossRef Link)

[28] Hongtao Xie, Yongdong Zhang, Jianlong Tan, Li Guo, Jintao Li, “Contextual Query Expansion for Image Retrieval,” *IEEE Trans on Multimedia*, vol. 16, no. 4, 2014. Article (CrossRef Link)

Yanping Ma received her Ph.D in Computer Application Technology from Ocean University of China in 2015. she is a lecturer at the Department of Information and Electrical Engineering in Ludong University. Her research interests include image processing, wavelet image compression, content-based image retrieval, community-based video recommendation .

Hailin Zou is a professor and the dean of Department Information and Electrical Engineering of LuDong University, YanTai, China. He received the Ph.D in Electrical Engineering from China University of Mining & Technology, Beijing. His major research interests include wavelets theory and its applications, ground penetrating radar signal processing, data fusion technology and intelligent processing systems.

Hongtao Xie received his Ph.D in Computer Application Technology from the Institute of Computing Technology, Chinese Academy of Sciences, China, in 2012. He is an associate professor in the Institute of Information Engineering, Chinese Academy of Sciences, China. His research interests include multimedia content analysis and retrieval, similarity search and parallel computing.

Qingtang Su received the Master Degree of Engineering from the School of Information and Electronic Engineering, Kunming University of Science and Technology, Kunming, China, in 2005, and is currently pursuing the Ph.D. degree in the School of Information Science and Engineering, East China University of Science and Technology, Shanghai, China. He is working as an associate professor in Department Information and Electric Engineering of LuDong University, Yantai, China. His research interests include image processing and information hiding.