Neurocomputing 151 (2015) 719–728

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Adaptive bit allocation hashing for approximate nearest neighbor search

Qin-Zhen Guo*, Zhi Zeng, Shuwu Zhang

Institute of Automation, Chinese Academy of Sciences, 95 Zhongguancun East Road, 100190 Beijing, China

ARTICLE INFO

Article history:
Received 17 March 2014
Received in revised form
18 September 2014
Accepted 6 October 2014
Available online 29 October 2014

Keywords:
Hashing
Adaptive bit allocation
Approximate nearest neighbor search
Hamming embedding
Image retrieval

ABSTRACT

Using hashing algorithms to learn binary codes representation of data for fast approximate nearest neighbor (ANN) search has attracted more and more attention. Most existing hashing methods employ various hash functions to encode data. The resulting binary codes can be obtained by concatenating bits produced by those hash functions. These methods usually have two main steps: projection and thresholding. One problem with these methods is that every dimension of the projected data is regarded as of same importance and encoded by one bit, which may result in ineffective codes. In this paper, we introduce an adaptive bit allocation hashing (ABAH) method to encode data for ANN search. The basic idea is, according to the dispersions of all the dimensions after projection we use different numbers of bits to encode them. In our method, more bits will be adaptively allocated to encode dimensions with larger dispersion while fewer bits for dimensions with smaller dispersion. This novel bit allocation scheme makes our hashing method effectively preserve the neighborhood structure in the original data space. Extensive experiments show that the proposed ABAH significantly outperforms other state-of-the-art methods for ANN search task.

1. Introduction

Nearest neighbor (NN) search has been widely used in computer vision applications, like multimedia retrieval, classification, annotation and other related application areas. The basic task of NN search is to find the samples that are most similar to a given query within a large database. Exhaustively comparing the query with each sample in the database is infeasible because the linear complexity is not scalable in practical situations such as large scale content-based image retrieval (CBIR). How to perform a fast NN search at large scale datasets has become an urgent research issue. Traditional linear search for NN requires scanning all the data (mostly vectors) in a dataset and the time complexity is $O(nd)$, where n is the size of the dataset and d is the dimension of vectors. Hence, it is computationally prohibitive to adopt linear search for massive datasets which might contain millions or even billions of vectors, especially when the vectors are high-dimensional, like the 128-dimensional SIFT descriptor [1]. Another problem of NN search in large scale datasets is the excessive, unacceptable storage consumption if traditional data formats are used.

In many applications like image retrieval, however, it is sufficient to return approximate nearest neighbor (ANN). The key idea of ANN is to find the NN with high probability instead of probability one. Several ANN search techniques have been developed including tree-based methods and hashing-based methods. Since the tree-based methods [2] have turned out to be not more efficient than exhaustive search for high dimensions, hashing-based ANN techniques [3–10] which aim to embed the data into Hamming space have attracted more and more attention. Specifically speaking, each vector is encoded as a binary code in the Hamming space and for preserving the neighborhood structure in the original data space, similar points in the original data space should be mapped to similar points in the Hamming space. Searching for similar neighbors is accomplished simply by finding the vectors that have codes within a small Hamming distance of the query's code. One of the advantages of the hashing methods is that the Hamming distance between two codes can be efficiently computed by the XOR operator followed by bit-count. Moreover, the storage will be largely reduced for storing the binary codes.

To generate a c-bit binary code, the hashing methods usually need c hash functions, and each hash function produces one bit of the binary code in two steps, projection and thresholding. The resulting binary code can be got by concatenating bits produced by those hash functions. More specifically,

$$H_i(x) = \begin{cases} 1 & u_i(x) > t_i \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

* Corresponding author. Tel.: +86 18810461353.
E-mail addresses: qinzhen.guo@ia.ac.cn (Q.-Z. Guo), zhi.zeng@ia.ac.cn (Z. Zeng), shuwu.zhang@ia.ac.cn (S. Zhang).

http://dx.doi.org/10.1016/j.neucom.2014.10.042
0925-2312/

$H_i(\boldsymbol{x})$ is the i-th hash function, u_i is the i-th projection function corresponding to $H_i(\boldsymbol{x})$, t_i is the corresponding threshold, and $\boldsymbol{x}$ is the original data. In the projection step, the projection functions are used to compute the projected real value $u_i(\boldsymbol{x})$. In the thresholding step, the i-th hash bit of $\boldsymbol{x}$ will be one if $u_i(\boldsymbol{x}) > t_i$. Otherwise, it will be zero. Here, the threshold, t_i, is typically set to zero if the data have zero mean. The final binary code is $(H_1(\boldsymbol{x}), H_2(\boldsymbol{x}),\ldots, H_c(\boldsymbol{x}))$, if we use c bits to encode the vectors.

However, the dispersions of all the dimensions after the projection are not identical and using one bit to encode each dimension will lead to insufficiency. For example, principal component analysis (PCA) is a popular projection method to project the original data into several dimensions of real values. And each of these projected dimensions can be quantized into one bit (zero or one) by thresholding, like PCAH [27]. But, the dispersions of different projected dimensions are different after PCA. Using the same number of bits for different projected dimensions is unreasonable because larger-dispersion dimensions will carry more information.

As illustrated in Fig. 1, the projection values of the data on direction w_1 have a larger dispersion and are dominant for Euclidean distance computation. To tackle the problem of the anisotropy of the projected data, in this paper, an adaptive bit allocation hashing (ABAH) approach is proposed to adaptively allocate different numbers of bits to encode every dimension for the ANN search. In our method, more bits to encode dimensions with larger dispersion while fewer bits will be allocated for dimensions with smaller dispersion for preserving the neighborhood structure. The main contributions of this work are outlined as follows:

- We prove that the variance of the p-th dimension is equivalent to $E((x_{ip}-x_{jp})^2)$ and a good way to measure dispersion under the assumption that data are independent and identically distributed (i.i.d.). x_{ip} is the p-th dimension of the i-th data and x_{jp} is the p-th dimension of the j-th data. Details can be seen in Section 3.1.2.
- A simple, intuitive, but effective and efficient, hashing method which we call adaptive bit allocation hashing (ABAH) is proposed. In this method, more bits are used adaptively to encode such dimensions with larger dispersion while fewer bits are used to encode the dimensions with smaller dispersion.
- Based on the naive ABAH scheme, we propose an improved version of ABAH. Extensive experiments have verified the superiority of our method.

This paper is an extended version of the work initially published in ICME 2013 [37]. Novel contributions over [37] include the improvement of ABAH and the application to image classification.

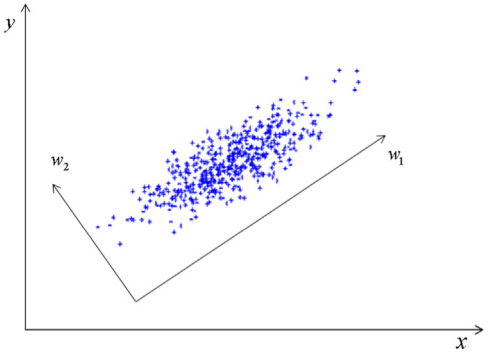

Fig. 1. Illustration of different dimensions' effect on Euclidean distance. Obviously, after projection, the values on direction w_1 have a larger dispersion and are dominant for Euclidean distance computation.

The rest of the paper is organized as follows. In Section 2, we introduce the related work. We describe our method in detail in Section 3. An improved version of our naive method is proposed in Section 4. Experimental results are presented in Section 5. The paper is concluded in Section 6.

2. Related work

There has been extensive research on approximate nearest neighbor (ANN) search since its great importance in many applications. Some tree-based methods such as modified KD-tree [11], spill tree [12] and vantage-point tree can be used for ANN search in a low-dimensional space. These methods usually partition the data space recursively to implement a similarity search. However, the tree-based methods can degenerate to a linear scan in the worst case where the number of the dimensions can be hundreds or even thousands. Hashing-based techniques are promising ANN search methods in terms of speed or storage and widely used in a large variety of applications.

In recent years, a lot of hashing algorithms have been developed. The pioneering work locality-sensitive hashing (LSH) [4] algorithm typically guarantees the probability for any two samples to fall into the same bucket. One popular method in LSH is to generate random projections from a particular probabilistic distribution [13]. However, since the random projections are data-independent, LSH may lead to quite inefficient codes in practice. Shift invariant kernel hashing (SIKH) [15] adopts projection functions in a way similar to LSH, but in SIKH, hash values are generated by a shifted cosine function. Restricted Boltzmann Machines (RBMs) [16] use multi-layer network to learn bits. Boosting Similarity Sensitive Coding (BoostSSC) [17] uses AdaBoost to classify a pair of input data as similar or nonsimilar. Spectral hashing (SH) [14], based on spectral graph partitioning, calculates the bits by thresholding a subset of eigenvectors of the Laplacian of the similarity graph and it has demonstrated significant improvements over LSH, RBMs and BoostSSC. As an extension of SH, hypergraph hashing [18] uses hyperplane to model the high-order relationships between social images. Active hashing [19] tries to select the most informative labels to learn hash functions. In order to effectively balance the precision and recall, complementary hashing [20] adopts multiple complementary hash tables which are learnt sequentially in a boosting manner. Semi-supervised hashing (SSH) [21] uses both labeled data and unlabeled data to learn hash functions. SPICA [22] finds independent projections by jointly optimizing both accuracy and time. Binary reconstruction embedding (BRE) [23] minimizes the reconstruction error between the distances in the original feature space and the Hamming distances of the corresponding binary codes to learn the hash functions. Minimal loss hashing (MLH) [24] formulates the hashing problem as a structured prediction problem that is based on the latent structural SVM framework. Self-Taught Hashing (STH) [9] which is an extension of SH, learns the hash functions via SVM for the unseen data points. LDAH [25] adopts Linear Discriminant Analysis (LDA) to learn the projection matrix and chooses the threshold by optimizing a loss function. Iterative quantization (ITQ) [38] introduces a procrustean approach that minimizes quantization loss to learn the projection matrix. In order to compute c-bit hash codes, PCAH [27] projects data to the c principal components, and then uses average values to binarize the coefficients. Most hashing methods regard every dimension of vectors as of same importance and allocate one bit to represent each dimension. Different from this, Anchor Graph Hashing (AGH) [28] uses a two-layer hash function to quantize every dimension. But in AGH, every dimension is still encoded by the same number of bits.

Most hashing-based methods first project the data and then use one bit to binarize them. However, the dispersions of the dimensions of the data after projection are not identical and using the same number of bits to binarize them is inappropriate. H. Jégou [26] deals with this problem in a simple way. In [26], a random orthogonal matrix which is got by QR factorizing a random matrix of Gaussian values is used to project the data to balance the dispersions of different dimensions. A new work isotropic hashing (IsoH) [29] also notices that different dimensions of the data after projection such as principal component analysis (PCA) have different dispersions and it is improper to use the same number of bits for different projected dimensions. In [29], IsoH learns an orthogonal matrix $\boldsymbol{Q}$ to re-project the PCA-projected data to guarantee the same dispersions of the different dimensions of the projected data.

In this paper, we tackle this problem in a novel perspective. According to the dispersion of each dimension, we adaptively allocate different numbers of bits to encode them. Our method can preserve the neighborhood structure of the original data. The details of our method will be stated in the next section.

A comparison of some methods on projection and thresholding is shown in Table 1.

3. Adaptive bit allocation hashing

First of all, let us introduce some notations. We have a training set of n data $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$, $\boldsymbol{x}_i \in R^d$, that form the rows of the data matrix $\boldsymbol{X} \in R^{n\times d}$. A binary code corresponding to each data point $\boldsymbol{x}_i$ is defined by $H_{ABAH}(\boldsymbol{x}_i)=\{0,1\}^c$, where c is the total code length. Since our method adaptively allocates different numbers of bits to encode different dimensions, the subcode of the p-th dimension is represented by s_p and the code length of s_p is c_p.

3.1. ABAH scheme

In this subsection, we introduce the details of our algorithm. Firstly, we use a projection to preprocess the original data to find the principal components. Secondly, in order to determine which dimension is more important after projection, every dimension's dispersion is calculated. Then according to the dispersion of each dimension, different numbers of bits are allocated to encode the corresponding dimension.

3.1.1. Projection

We adopt PCA to project the original data for two reasons. One is that the dimensions of vectors after PCA projection are irrelevant and the first dimensions are the principal components which can better describe the original data. The other is that the eigenvalues used in PCA algorithm serve as a measurement of dispersion (see Section 3.1.2) and can be easily used to compute every component's code length (see Section 3.1.3).

It is important to note that all the data discussed below are PCA-projected data and we just use PCA to rotate the data and not to reduce dimensionality in our method.

3.1.2. Dispersion measurement

Given $\boldsymbol{x}_i=(x_{i1}, x_{i2}, \ldots, x_{id})$ and $\boldsymbol{x}_j=(x_{j1}, x_{j2}, \ldots, x_{jd})$, which are two randomly chosen d-dimensional data, the Euclidean distance between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, $D(\boldsymbol{x}_i, \boldsymbol{x}_j)=\mathrm{sqrt}((x_{i1}-x_{j1})^2+(x_{i2}-x_{j2})^2+\ldots+(x_{id}-x_{jd})^2)$ is mainly determined by those dimensions that have a larger value of $(x_{ip}-x_{jp})^2$ for $1\le p\le d$. Hence, if the data on some dimensions have a larger $E((x_{ip}-x_{jp})^2)$, which is related to the dispersion of the p-th dimension, those dimensions will be more important for computing the Euclidean distance. When embedding the data into Hamming space, in order to maintain the neighborhood structure, important dimensions in the Euclidean space should also be important in the Hamming space. If we use the same numbers of bits to encode every dimension, the information loss of dimensions with larger dispersion caused by thresholding will increase. And it will be superfluous for dimensions with less dispersion.

As proved below, the variance of the p-the dimension is equivalent to $E((x_{ip}-x_{jp})^2)$ under the assumption that the data are independent and identically distributed (i.i.d.). For the p-th column of $\boldsymbol{X}$,

$$\begin{aligned} E((x_{ip}-x_{jp})^2) &= E(x_{ip}^2+x_{jp}^2-2x_{ip}x_{jp}) \\ &= 2Ex_{ip}^2-2Ex_{ip}x_{jp} \end{aligned} \tag{2}$$

Assume the training data are i.i.d., and then we have

$$\begin{aligned} E((x_{ip}-x_{jp})^2) &= 2Ex_{ip}^2-2Ex_{ip}x_{jp} \\ &= 2Ex_{ip}^2-2Ex_{ip}Ex_{jp} \\ &= 2Ex_{ip}^2-2{\mu_p}^2 \end{aligned} \tag{3}$$

The variance of the p-th column of $\boldsymbol{X}$ is

$$\begin{aligned} Var_p(X) &= E((x_{ip}-\mu_p)^2) \\ &= E(x_{ip}^2+{\mu_p}^2-2x_{ip}\mu_p) \\ &= Ex_{ip}^2+{\mu_p}^2-2\mu_pEx_{ip} \\ &= Ex_{ip}^2-{\mu_p}^2 \end{aligned} \tag{4}$$

Table 1
Comparison of different methods on projection and thresholding.

Methods	Projection	Thresholding	Objective function
LSH [13]	$\boldsymbol{XP}$	$\mathrm{Sign}(\boldsymbol{XP})$	$\boldsymbol{P}$ is a random matrix
PCAH [27]	$\boldsymbol{XP}$	$\mathrm{Sign}(\boldsymbol{XP}+\boldsymbol{T})$	$[\Lambda,\boldsymbol{P}]=\mathrm{PCA}(\boldsymbol{X})$
HE [26]	$\boldsymbol{XP}$	$\mathrm{Sign}(\boldsymbol{XP}+\boldsymbol{T})$	$[\boldsymbol{P},\boldsymbol{R}]=\mathrm{qr}(\boldsymbol{M})$
SIKH [15]	$\boldsymbol{XP}$	$\mathrm{Sign}(\cos(\boldsymbol{XP}+\boldsymbol{T}))$	$\boldsymbol{P}\sim \mathrm{P_K}$
MLH [24]	$\boldsymbol{XP}$	$\mathrm{Sign}(\boldsymbol{XP})$	$\arg\min L(H(x_i), H(x_j), s_{ij})$
IsoH [29]	$\boldsymbol{XP}$	$\mathrm{Sign}(\boldsymbol{XP})$	$\arg\min \|T-Z\|_F$
LDAH [25]	$\boldsymbol{XP}$	$\mathrm{Sign}(\boldsymbol{XP}+\boldsymbol{T})$	$\arg\min_P \alpha tr\{P\Sigma_P P^T\}-tr\{P\Sigma_N P^T\}$
ITQ [38]	$\boldsymbol{XP}$	$\mathrm{Sign}(\boldsymbol{XP})$	$\arg\min_{B,P}\|B-VP\|_F^2$
STH [9]	$\boldsymbol{XP}$	$\mathrm{Sign}(\boldsymbol{XP})$	$\arg\min_{p_i}\frac{1}{2}p_i^T p_i+\frac{C}{n}\sum\xi_i$
SPICA [22]	$\boldsymbol{XP}$	$\mathrm{Sign}(\boldsymbol{XP}+\boldsymbol{T})$	$\arg\max_{p_i}\sum\|g_0-\frac{1}{N}\sum_j(G(p_i x_j))\|^2$
BRE [23]	$\sum_{q=1}^{s}(W_{pq}\kappa(x_{pq},x))$	$sign(\sum_{q=1}^{s}(W_{pq}\kappa(x_{pq},x)))$	$\arg\min\sum(d(x_i-x_j)-d^{\sim}(x_i-x_j))^2$

$\boldsymbol{X}$ is a matrix whose one row is a data (or vector). $\boldsymbol{P}$ is the projection matrix and $\boldsymbol{p_i}$ is the i-th column of $\boldsymbol{P}$. $\boldsymbol{P}$ can be obtained by solving the objective function. $\boldsymbol{T}$ is the threshold. The details of different methods can be seen in corresponding references.

From the derivation, we can see that the variance is equivalent to $E((x_{ip}-x_{jp})^2)$. $\mu_p$is the mean of the p-th column of $\boldsymbol{X}$. So we use variance to measure dispersion.

Besides, the variance of each dimension is easy to be obtained since we use PCA projection to project the data, and the eigenvalue associated with a dimension indicates the variance of this dimension. So we use variance to measure dispersion. If one dimension has a larger variance, more bits will be allocated to encode this dimension, otherwise fewer bits will be allocated.

3.1.3. Calculating code lengths

To maintain the neighborhood structure, we use more bits to encode those dimensions that have larger variance. In other words, if the variance of the p-th column of the PCA projected data is larger, more bits will be adaptively used to encode the p-th dimension.

Simply and intuitively, given total code length c, the p-th dimension's code length c_p will be

$$c_p = \left[c \cdot \frac{\lambda_p}{\Sigma_{i=1}^{d} \lambda_i} + 0.5 \right] \tag{5}$$

λ_p is the variance of the p-th dimension of the PCA projected data for $1 \le p \le d$ and [x] means floor(x). One problem with Eq. (5) is that if some λ_p are very close to each other and they are not large enough, e.g., $\lambda_p < 0.5(1/c)\Sigma_{i=1}^{d}\lambda_i$, this will result in those $c_p=0$ and$\Sigma_{i=1}^{d} c_i \neq c$. To handle this problem, we use a variant strategy as Eq. (6). If $c_p=0$ according to Eq. (6), and $\Sigma_{t=1}^{p-1} c_t \neq c$, we set $c_p=1$. Since PCA will find the principal components with degressive variance, the subcode of the p-th dimension is longer than that of the $(p+1)$-th dimension for $1 \le p \le (d-1)$.

$$c_p = \begin{cases} \left[c \cdot \frac{\lambda_p}{\Sigma_{i=p}^{d} \lambda_i} + 0.5 \right] & p = 1 \\ \left[(c - \Sigma_{t=1}^{p-1} c_t) \cdot \frac{\lambda_p}{\Sigma_{i=p}^{d} \lambda_i} + 0.5 \right] & p \ge 2 \end{cases} \tag{6}$$

3.1.4. Encoding every dimension

If two vectors are close to each other, the Hamming distance between their binary codes should be small. We achieve this goal by making every dimension's subcode close to each other for similar data. Details are illustrated as follows.

For a training vector $\boldsymbol{x}_i=(x_{i1}, x_{i2}, \ldots, x_{id})$, c_p bits are allocated to encode the p-th dimension. Firstly, we divide the range of the p-th dimension into c_p+1 parts by c_p thresholds. The c_p thresholds are chosen in a simple way

$$t_j^p = x_p^{\min} + (j/(c_p+1)) \cdot (x_p^{\max} - x_p^{\min}) \tag{7}$$

where $t_j^p$is the j-th threshold of the p-th dimension, for $1 \le j \le c_p$ and $x_p^{\min}$and$x_p^{\max}$ indicate the minimum value and maximum value of the p-th column of the training matrix $\boldsymbol{X}$. If$t_{j-1}^p \le x_{ip} \le t_j^p$, for $2 \le j \le c_p$, we encode x_{ip} with subcode s_p which consists of (c_p-j+1) zeros followed by $(j-1)$ ones. If$x_{ip} > t_{c_p}^p$, s_p will consist of c_p ones. If$x_{ip} < t_1^p$, s_p will consist of c_p zeros. In this way, the Hamming distance between two subcodes corresponding to two values that are close to each other will be small. And the final binary code $H_{ABAH}(\boldsymbol{x}_i)$ of $\boldsymbol{x}_i$ can be obtained by concatenating all the subcodes, i.e., $H_{ABAH}(\boldsymbol{x}_i)=(s_1, s_2, \ldots, s_t)$, subject to $\Sigma_{p=1}^{t} c_p = c$.

We term this simple algorithm ABAH_UN. ABAH_UN has one limitation that the thresholds are chosen uniformly from the range of each dimension. This may divide two values close to each other into two different regions. We have experimented with the k-means method to find every dimension's 'thresholds.' Specifically speaking, for the p-th dimension we use one-dimensional k-means algorithm to find (c_p+1) centroids r_f $(f=1,\ldots, c_p+1)$ with ascending order. Values belonging to the f-th cluster will be encoded by subcode s_p which consists of (c_p-f+1) zeros followed by $(f-1)$ ones. The k-means algorithm can give a better partition and the centroids are better representation of every part than ABAH_UN. It achieves better results and we name it ABAH_KM.

3.2. Summary of ABAH procedure

Given a training set $\{\boldsymbol{x}_i\}$ and a desired code length c, the whole simple learning procedure of ABAH can be summarized as follows:

- Finding the principal components of the original data by PCA.
- Calculating the code length of every dimension in the way described in Section 3.1.3.
- Calculating every dimension's thresholds in the way described in Section 3.1.4.

For the data (query and database) to be encoded, we use the trained code lengths and the thresholds to encode the data one dimension by one dimension and then concatenate on the subcodes to obtain the final binary code in the way described in Section 3.1.4.

3.3. Discussion

Now let us discuss the time complexity of ABAH. The time complexity of PCA is $O(nd^2)$ since $d \ll n$ in the projection phase, and that of code lengths calculating step is $O(d)$ (see Section 3.1.3). Once we get the code length of every dimension, the subcodes can be calculated and stored in a table before converting data to binary codes. Using thresholds to encode every dimension by subcodes can be achieved very fast by looking up the table. For ABAH_UN, to find the thresholds, in the worst case, the time consuming is $O(nc)$. Since our methods can generate codes longer than data dimension, if c is larger than data dimension d, the time consumption of computing thresholds will be $O(nd+c)$. For ABAH_KM, the time complexity of finding the thresholds is $O(nct)$ where t is the iterations. But since PCA will find the principal components and more bits will be allocated to those components, in reality, the thresholds computing time is far less than $O(nc)$ for ABAH_UN. Hence, the main time complexity of ABAH is from the PCA phase.

4. Improved adaptive bit allocation hashing

The naive ABAH method (both ABAH_UN and ABAH_KM) have two problems. One problem with the naive ABAH is that when encoding every dimension, the last dimensions are not used. In other words, these dimensions' code lengths are zero. That is to say, we use those dimensions to calculate code lengths, but we do not encode those dimensions, which may incur insufficiency. Besides, when using PCA, only selecting projected dimensions with large eigenvalues has been demonstrated to be sufficient

Algorithm 1. Choosing dimensions

1. **Input:** a training set $\boldsymbol{X}$, the bit number c
2. **Output:** the number of used dimensions p
3. **Initialize:** $p=d$, $[\Lambda, W]=\mathrm{PCA}(\boldsymbol{X}, d)$, a d-dim array $\boldsymbol{a}$ whose i-th element is $\Lambda(i,i)$.Λ is a $d \times d$ matrix, whose diagonal values are eigenvalues.
4. **Repeat**
   Use the first p elements of $\boldsymbol{a}$ to calculate code lengths, as stated in Section 3.1.3.
   Update: set p as the number of dimensions whose code lengths are not zero. $p=i$, s. t. $c_i > 0$, $c_{i+1}=0$.
5. **Until** convergence.

enough to preserve the neighborhood structure of the data. In order to solve this problem, we iteratively discard the dimensions whose code length is zero. Then we use the remaining dimensions to recalculate code lengths. The iteration termination requirement is that the code length of the last dimension of the used dimensions is one (see Algorithm 1). Thus, we can maximally use the dimensions and all the used dimensions can be encoded by at least one bit.

The other problem is that the proposed method in Eq. 6 is not an optimal solution. If the first p-1 components use less than c bits, the p-th component which may not be large enough to use one bit will be assigned one. Besides, if the last few dimensions' variances are almost equal, which are very common after PCA projection, more bits may be allocated to the dimension with smaller variance (see Fig. 2(a)). For example, assuming there are four bits left (corresponding to $(c-\Sigma_{t=1}^{p-1}c_t)$in Eq. (6)), and three dimensions with variance 1.000, 0.840, and 0.830, respectively, to be encoded, according to Eq. (6), the subcode lengths are 1, 2, and 1, which will result in allocating more bits to the dimension with smaller variance.

We use a simple approach to deal with this problem. After calculating the code lengths using Eq. (6), we sort the code lengths with descending order. And then, the sorted code lengths are allocated to corresponding dimensions. This can guarantee that dimensions with bigger variance will not be allocated fewer bits than dimensions with smaller variance.

For these two improvements, we evaluate them separately and find that the second improvement which slightly improves the results is trivial. But the first improvement can largely boost the results. We abbreviate the improved ABAH_KM as ABAH_IM (improved by Algorithm 1 and sorting the code lengths). The subcode lengths' comparison of ABAH_KM and ABAH_IM can be seen in Fig. 2. Please note that the first 76 dimensions' subcode lengths of ABAH_KM are got by training all the dimensions of the training data, i.e., 128-dim. But the first 55 dimensions' subcode lengths of ABAH_IM are obtained only by training the first 55 dimensions of the training data. Besides, though the ninth dimension has a larger dispersion (or variance) than the tenth dimension, the naive ABAH allocates four bits for the ninth dimension, while it allocates five bits for the tenth dimension (see Fig. 2(a)), which will lead to ineffectiveness.

4.1. Connection to source coding

In source coding area, there is an optimal bit allocation for encoding Gaussian distributed data [41] and other distributed data [42,43]. However, our hashing method and source coding have different objectives. In source coding, on one hand, the objective is to minimize the quantization distortion between the original data and the reconstructed data (quantizer outputs) with some constraints (resolution or entropy). On the other hand, in source coding, the binary codes are just used to index the quantizer outputs (codebook) for transmitting through the channel, and the index does not reflect the similarity or dissimilarity of the indexed data (quantizer outputs). In our hashing method, our objective is to convert the original data into binary codes (index) while the index can keep the similarity or dissimilarity of the indexed data. We evaluate the bit allocation scheme in [41] with our hashing method and the results are presented in Section 5.5. Though the bit allocation scheme in our method may not be optimal, the gains made in search performance provide excellent justification for this loss in optimality. We do extensive experiments to compare with the state-of-the-art and the comparison results provide experimental justification of our method.

5. Experiments

5.1. Datasets

For ANN search task, we perform experiments with the following two datasets:

- ANN-GIST-1M [30]: A set of 960 dimensional, one million GIST descriptors.
- ANN-SIFT-1M [30]: A set of 128 dimensional, one million SIFT descriptors.

For ANN-GIST-1M and ANN-SIFT-1M, we randomly choose 1 K vectors as queries, 100 K of the rest as training set, and the remaining as database. The ground truth is defined by the 1000 nearest neighbors computed by the exhaustive, linear scan based on the Euclidean distance. The retrieved points are computed by ranking their Hamming distance to the query. The performance is measured by mean average precision (mAP) and recall, which are defined as follows:

$$\text{Recall} = \frac{\text{the number of retrieved relevant points}}{\text{the number of all relevant points}} \tag{8}$$

$$\text{Precision} = \frac{\text{the number of retrieved relevant points}}{\text{the number of all retrieved points}} \tag{9}$$

$$\text{mAP} = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{m_i}\sum_{j=1}^{m_i} Precision(R_{ij}) \tag{10}$$

where Q is a query set, $|Q|$ is the cardinality of Q, m_i is the number of points relevant to q_i ($q_i \in Q$), R_{ij} is the set of the first j retrieval results. $Precision(R_{ij})$ means the proportion of the retrieved relevant points in R_{ij}.

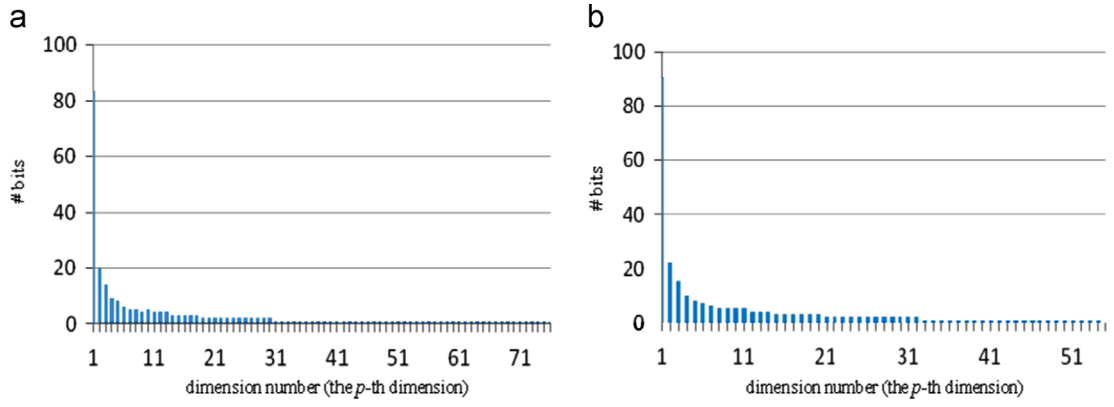

Fig. 2. Subcode lengths of ABAH_KM and ABAH_IM on ANN-SIFT-1M while the total bit number c=256. (a) subcode lengths of ABAH_KM and (b) subcode lengths of ABAH_IM.

Finally, we employ our methods for practical application for image retrieval on the image set MIRFLICKR-1M [31] and UKB [32] and image classification on CIFAR-10 [33].

- MIRFLICKR-1M: An image set, with various kinds of contents, containing one million images downloaded from the Flickr website. And the sizes of images in this set are not identical.
- UKB: The University of Kentucky Benchmark (UKB) contains 10,200 images of 2550 objects, 4 pictures correspond to each object, taken from different angles. The size of all the images is 640×480.
- CIFAR-10: As a subset of the well-known 80 M tiny image collection [34], CIFAR-10 consists of 60,000 32×32 color images which are manually labeled as ten classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck) with 6000 samples for each class.

We represent each image in all image sets with a 512-dimensional GIST descriptor [35]. For MIRFLICKR-1M, the image set is randomly divided into three parts: learning (100 K), database (899 K), and query (1 K). The ground truth is defined by exact NN search and we also use mAP and recall to measure the performance. For image set UKB, each image is used in turn as query to search through the 10,200 images. The accuracy is measured in terms of the number of relevant images retrieved in the top 4, i.e., $4 \times$ precision@4. For learning purpose on UKB, we use an independent image set (the MIRFLICKR-1M training set) to learn the parameters (code lengths and thresholds of each dimension). As for CIFAR-10, we use the default training set consisting of 50,000 images and test set with 10,000 images. The performance is measured by classification precision (precision, for simplicity). One important thing we should know is that the retrieval and classification results not only depend on the hashing method, but are also affected by the image representation to a great extent. We evaluate ABAH on these datasets just for comparing with other hashing methods.

For all the experiments, we use a machine consisting of i7-2600 CPU and 16 GB main memory. Note that we implement linear scan in the query phase where we compute the Hamming distance between the query's code and all the codes in the database. For fast approximate NN search in Hamming space, we can use a simple approach to achieve sub-linear search by using the hash-table method in [4]. Also, we can use the delicate inverted file method described in [39] to achieve sub-linear search. However, since the Hamming distance computation is extremely fast (see Table 5), we still employ linear scan in the experiments.

5.2. Compared methods

- LSH: LSH employs simple random projections [13].
- SH: Spectral Hashing [14].
- PCAH: PCAH [27] uses the average value of each dimension to binarize this dimension with one bit after PCA projection.
- HE: Use a random orthogonal matrix to project the data [26].
- IsoH: Use a learned orthogonal matrix to project the PCA-projected data to guarantee the same variance of every dimension [29].
- ITQ: Adopt a projection matrix to minimize quantization loss [38].
- ABAH_UN: The thresholds in this method are obtained uniformly.
- ABAH_KM: An improved version of our ABAH with the thresholds obtained by k-means algorithm as stated in Section 3.1.4.
- ABAH_IM: The improved ABAH_KM as stated in Section 4.

5.3. Results

Fig. 3 shows the mAP of returning the first 1000 nearest neighbors of all the tested methods on ANN-GIST-1M and ANN-SIFT-1M. Since SH, PCAH, HE, IsoH, and ITQ cannot generate codes longer than data dimension while ABAH can, we do not plot the results when codes are longer than 128 bits for SH, PCAH, HE, IsoH, ITQ on dataset ANN-SIFT-1M in Fig. 3(b). ABAH_IM performs better most of the time across the tested bit lengths ranging from 64 to 512 bits on both dataset especially when longer bits are used. Similar to that reported in [36], PCAH performs worse when using more bits to encode the data on ANN-GIST-1M. ITQ performs well when fewer bits are used to encode the data on ANN-GIST-1M. However, it achieves poor results on ANN-SIFT-1M. The random orthogonal projection method, HE, achieves good results, even better than the isotropic-guaranteed method IsoH, but not better than our ABAH_IM approach.

As we can see, ABAH_KM achieves constantly better results than ABAH_UN on the two datasets from 16 bits to 512 bits, especially when using more bits to encode the data. This is because the cluster centers got by k-means algorithm are better representations of the data and can give a better partition.

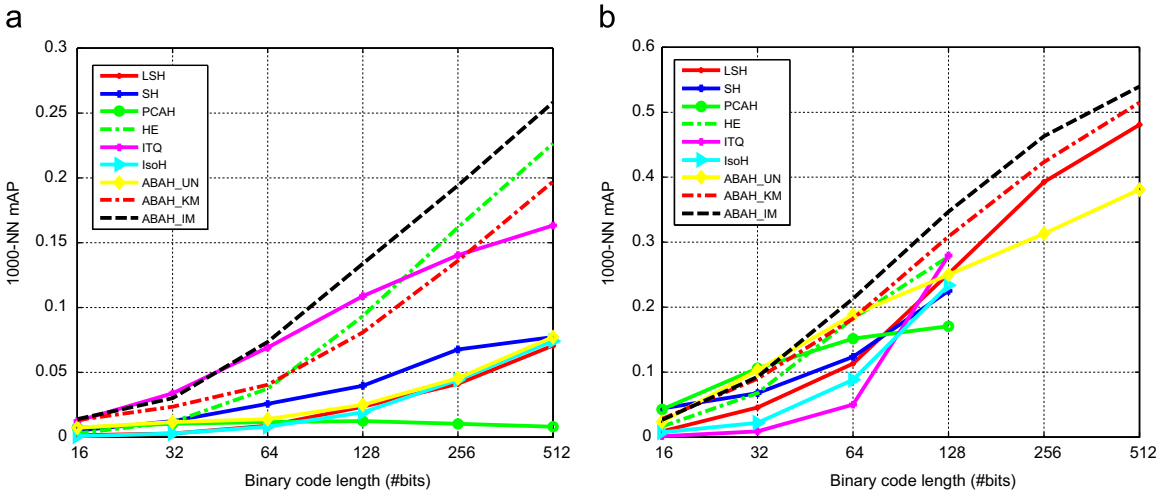

Fig. 3. Comparison of ABAH to state-of-the-art methods in terms of 1000-NN mAP: (a) experiment on ANN-GIST-1M dataset and (b) experiment on ANN-SIFT-1M dataset.

However, since the bit allocation algorithm in ABAH_KM is not optimal, it performs worse than ABAH_IM.

One phenomenon worthy of our attention is that a 16-bit code cannot represent one vector uniquely because 16-bit codes at most have 65,536 different representations, while we have far more vectors in the dataset which has 899 K vectors. Our ABAH methods also get significantly better results in this case on ANN-GIST-1M.

In Fig. 4, experimental results on dataset ANN-GIST-1M and ANN-SIFT-1M in terms of Recall@1000 are presented. By the same token as Fig. 3(b), we do not plot the results of 256-bit and 512-bit for SH, PCAH, HE, IsoH, and ITQ in Fig. 4(b). Both 1000-NN mAP and Recall@1000 on the two dataset almost have the same trend. We can see that our ABAH_IM method achieves the best results on ANN-GIST-1M. As shown in Fig. 4, with longer codes for ABAH, better results are obtained. One can strike a balance between accuracy and efficiency by choosing appropriate number of bits.

For image retrieval, we test our methods on image set MIRFLICKR-1M and UKB. The experiment results on MIRFLICKR-1M are presented in Fig. 5. The ABAH_IM hashing method performs well, especially when the bits are longer. ITQ achieves almost the same results as ABAH_IM with shorter codes. PCAH performs worst, because the hash hyperplanes of PCAH cannot capture the neighborhood structure information. Different from PCAH, IsoH that guarantees dimensions have isotropic variance performs better than PCAH when bits are longer than 64 bits. Since LSH adopts random hash functions which are independent of the training data, its result is also inferior. SH that relies upon its uniform data assumption also gets poor performance.

Image retrieval results on UKB can be seen in Fig. 6. Our ABAH_KM hashing approach outperforms other compared methods from 32 bits to 512 bits. PCAH achieves surprisingly good results with shorter codes in this image set. However, our ABAH_IM method does not achieve the best result. There are two reasons. One is that the ABAH_IM hashing method can better capture the neighborhood structure in terms of Euclidean distance, but the ground truth of UKB image set are semantic labels. The results indicate that data close to each other in Euclidean space may not be semantically related. The other is that the i.i.d. assumption may not hold on this dataset. NN Baseline is computed by the exhaustive, linear scan based on the Euclidean distance.

Finally, we test our methods for image classification task on image set CIFAR-10. The images are represented by 512-dim GIST descriptors. We use the default training set of 50,000 images to train the parameters of SH, PCAH, HE, ITQ, IsoH and our ABAH methods. Given a test sample, we efficiently return the first K nearest neighbors in Hamming space. Then we apply a K-nn classifier to classify the test sample. The experiment results are presented in Tables 2–4 for $K=1$, 10, and 20, respectively, in terms of classification precision. We also expand the experiment for K varying from 1 to 65 with 512 bits encoding the image descriptors (see Fig. 7). We can find that our ABAH methods are comparable, if not superior, to the state-of-the-art methods, such as ITQ. NN baseline is also computed by exhaustive search.

5.4. Runtime comparison

Table 5 gives the time comparisons between exact NN search and ABAH_IM. The results are obtained with our unoptimized C++ code. We use one query to linear scan the dataset whose size is 899,000 to find the nearest neighbor. For ABAH_IM, the time includes converting data to binary codes and exhaustive search in Hamming space.

5.5. Additional analysis

For comparison, we also use the following two methods to obtain the thresholds of each dimension. One is ABAH_SAX where we use the method described in [40] to model the data with single Gaussian distribution to get the thresholds that keep a constant mass in each cell. The other is ABAH_GMM. Based on the probability density function (pdf) of each dimension, in ABAH_GMM, we use Gaussian mixture model (GMM) to learn the centroids of each dimension.

Tables 6 and 7 show the experiment results on ANN-SIFT-1M for ABAH_IM, ABAH_SAX and ABAH_GMM. Note that these three methods are also improved by Algorithm 1 and sorting the code lengths as described in Section 4. ABAH_IM constantly achieves the best results. ABAH_SAX gets poor results. The possible reason is that each dimension of the data is not single Gaussian distribution, and using Gaussian lookup table to get the thresholds that keep a constant mass in each cell may not appropriately partition the data. ABAH_GMM where we model the data of each dimension with one-dimensional GMM achieves almost the same results as ABAH_IM. Actually, ABAH_IM is a simplified non-probabilistic version of ABAH_GMM. However, due to higher complexity of ABAH_GMM, we still use ABAH_IM to compare with other hashing methods. Besides, we also compare ABAH_IM and ABAH_GMM on ANN-GIST-1M dataset. Because of the relation between k-means and GMM, they still achieve almost the same results. Due to limited space, we do not present the results.

To evaluate the bit allocation scheme in [41] with our hashing method, we use the bit allocation scheme in [41] to allocate bits to

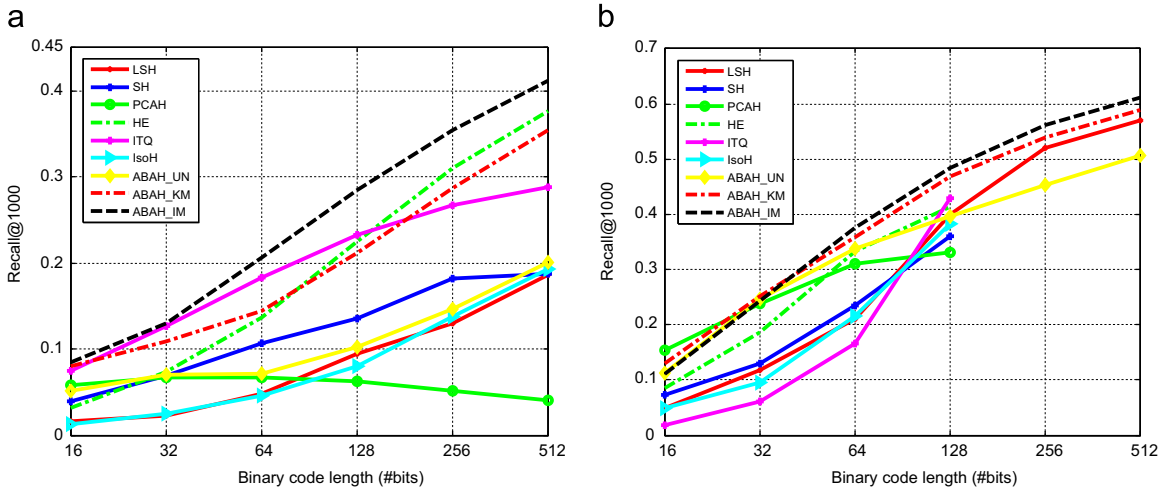

Fig. 4. Comparison of ABAH to state-of-the-art methods in terms of Recall@1000: (a) experiment on ANN-GIST-1M dataset and (b) experiment on ANN-SIFT-1M dataset.

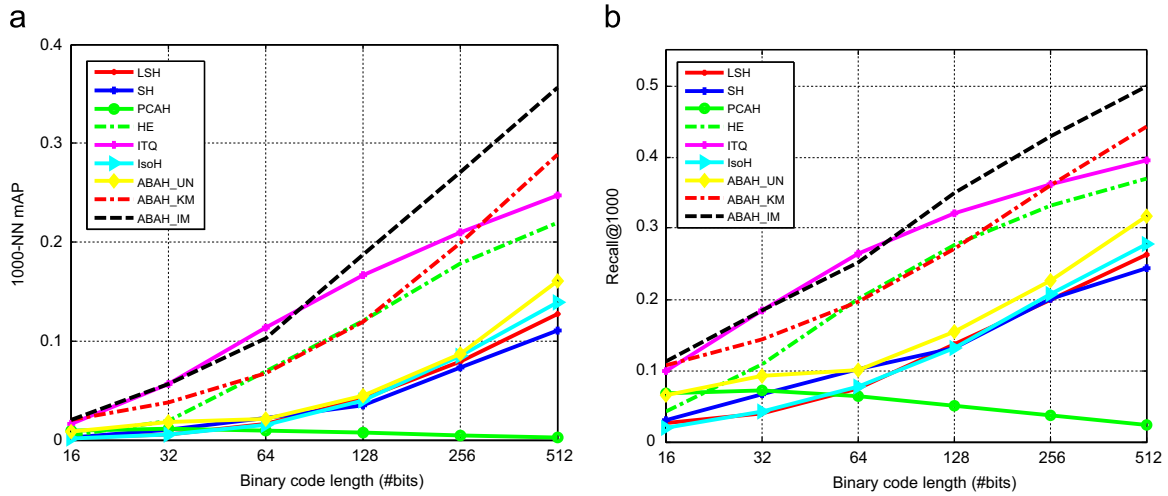

Fig. 5. Comparison of ABAH to state-of-the-art methods on MIRFLICKR-1M in terms of mAP and recall: (a) mAP on MIRFLICKR-1M image set and (b) recall on MIRFLICKR-1M image set.

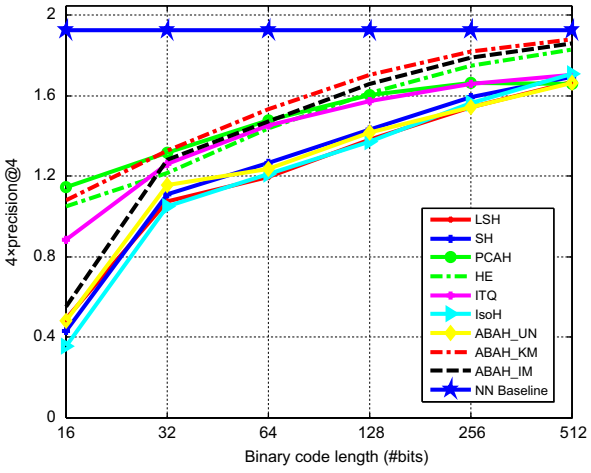

Fig. 6. Comparison of ABAH to state-of-the-art methods on UKB dataset.

Table 2
Comparison of hashing methods on CIFAR-10 in terms of precision for $K=1$.

# Bits	16	32	64	128	256	512
LSH	0.1448	0.2001	0.2570	0.3226	0.3882	0.4347
SH	0.2392	0.2902	0.3350	0.3724	0.3850	0.3564
PCAH	0.2627	0.3171	0.3581	0.3449	0.3342	0.3335
HE	0.2008	0.2850	0.3587	0.4113	0.4460	0.4767
ITQ	**0.2749**	0.3513	0.3671	0.4307	0.4505	**0.4992**
IsoH	0.1713	0.2065	0.2695	0.3232	0.3850	0.4434
ABAH_UN	0.2070	0.2926	0.3231	0.3622	0.3913	0.4124
ABAH_KM	0.2482	**0.3583**	0.3742	0.4325	0.4471	0.4649
ABAH_IM	0.2259	0.3458	**0.3833**	**0.4397**	**0.4645**	0.4946

each dimension of the data and adopt k-means (SC_IM) and GMM (SC_GMM), respectively, to learn the centriods of each dimension. In SC_IM and SC_GMM, we use the one-dim encoding scheme of our method to encode each dimension. The results are presented in Tables 8, and 9. Since the bit allocation scheme of our method is based on the consideration discussed in Section 3.1.2 while the bit allocation scheme in [41] is based on the objective in source coding, our method achieves better results than the source coding-based methods at all tested bit lengths.

We also evaluate ABAH_IM on the raw data and random projected data. For the raw and random data, we compute the variance of each dimension and rank the dimensions by their

Table 3
Comparison of hashing methods on CIFAR-10 in terms of precision for $K=10$.

# Bits	16	32	64	128	256	512
LSH	0.1926	0.2467	0.3194	0.3802	0.4473	0.4818
SH	0.2961	0.3540	0.4085	0.4437	0.4458	0.3884
PCAH	0.3411	0.4011	0.4260	0.4039	0.3971	0.3786
HE	0.2591	0.3477	0.4156	0.4764	0.5187	0.5117
ITQ	**0.3511**	0.4162	0.4546	0.4918	0.5127	0.5193
IsoH	0.1920	0.2586	0.3211	0.3790	0.4426	0.4941
ABAH_UN	0.2671	0.3625	0.4055	0.4331	0.4728	0.4733
ABAH_KM	0.3175	**0.4212**	0.4588	**0.5049**	0.5140	0.5156
ABAH_IM	0.3003	0.4012	**0.4625**	0.4973	**0.5208**	**0.5210**

Table 4
Comparison of hashing methods on CIFAR-10 in terms of precision for $K=20$.

# Bits	16	32	64	128	256	512
LSH	0.2057	0.2647	0.3380	0.3916	0.4586	0.4846
SH	0.3141	0.3821	0.4266	0.4703	0.4649	0.4112
PCAH	0.3604	0.4247	0.4639	0.4319	0.4293	0.4185
HE	0.2784	0.3634	0.4295	0.4841	0.5118	0.5186
ITQ	**0.3675**	**0.4327**	0.4693	0.4934	0.5142	0.5245
IsoH	0.2108	0.2678	0.3427	0.3917	0.4559	0.4991
ABAH_UN	0.2901	0.3766	0.4210	0.4498	0.4830	0.4891
ABAH_KM	0.3433	0.4314	**0.4852**	0.5232	0.5208	0.5202
ABAH_IM	0.3150	0.4147	0.4778	**0.5261**	**0.5291**	**0.5255**

variance with descending order. Then we allocate bits to each dimension with the same scheme of ABAH_IM. The results are presented in Tables 10 and 11. Our ABAH_IM method achieves the best results. The reason is that we cannot find the principle component of the data on both the raw data and random projected data as illustrated in Fig. 1. And it is hard to determine which dimension is important for computing Euclidean distance and also for computing Hamming distance. According to Eq. (6), many dimensions' code length is zero, which can be seen as dimension selection. As to PCA, we select some principal components. However, as to raw data and random projected data, the dimensions we select are not principal components which may be discarded.

5.6. Discussion

Different dimensions of the projected data have different variances and it is inappropriate to use the same number of bits

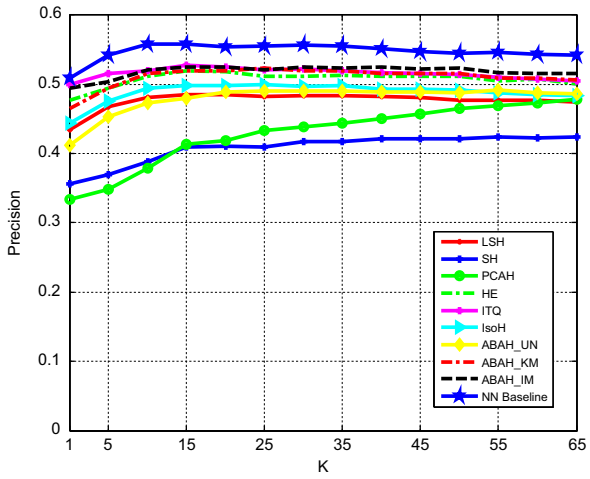

Fig. 7. Comparison of ABAH to state-of-the-art methods on CIFAR-10 image set for image classification task in terms of precision with 512 bits encoding the image descriptors.

Table 5
Time comparisons (ms) of exact NN search in Euclidean space on ANN-SIFT-1M and ANN-GIST-1M and ABAH_IM in Hamming space.

# Bits	16	32	64	128	256	512
ABAH_IM	< 1	< 1	< 1	< 1	15	31
Original space (128-dimensional SIFT)	437					
Original space (960-dimensional GIST)	2714					

Table 6
Comparison of methods to obtain thresholds using ANN-SIFT-1M in terms of mAP.

# Bits	16	32	64	128	256	512
ABAH_IM	**0.0254**	**0.0926**	**0.2129**	0.3471	0.4630	0.5403
ABAH_SAX	0.0100	0.0341	0.0993	0.1250	0.1247	0.0658
ABAH_GMM	0.0201	0.0915	0.2100	**0.3547**	**0.4724**	**0.5453**

Table 7
Comparison of methods to obtain thresholds using ANN-SIFT-1M in terms of Recall@1000.

# Bits	16	32	64	128	256	512
ABAH_ IM	**0.1104**	**0.2434**	**0.3750**	0.4835	0.5608	0.6116
ABAH_SAX	0.0550	0.1220	0.2427	0.2717	0.2664	0.1872
ABAH_GMM	0.1082	0.2412	0.3729	**0.4901**	**0.5692**	**0.6169**

Table 8
Comparison of bit allocation methods with different thresholds using ANN-SIFT-1M in terms of mAP.

# Bits	16	32	64	128	256	512
ABAH_IM	**0.0254**	**0.0926**	**0.2129**	0.3471	0.4630	0.5403
ABAH_GMM	0.0201	0.0915	0.2100	**0.3547**	**0.4724**	**0.5453**
SC_IM	0.0192	0.0859	0.1920	0.3075	0.4144	0.4693
SC _GMM	0.0188	0.0890	0.1958	0.3135	0.4166	0.4660

for different projected dimensions like PCAH. As far as we know, there are two methods to deal with this problem. One is adaptive bit allocation scheme like our ABAH methods. We use more bits to encode the dimensions with larger variance while fewer bits to encode dimensions with smaller variance. The other is that we project the data to a new space where every dimension of the data has the same variance like IsoH.

Table 9
Comparison of bit allocation methods with different thresholds using ANN-SIFT-1M in terms of Recall@1000.

# Bits	16	32	64	128	256	512
ABAH_IM	**0.1104**	**0.2434**	**0.3750**	0.4835	0.5608	0.6116
ABAH_GMM	0.1082	0.2412	0.3729	**0.4901**	**0.5692**	**0.6169**
SC_IM	0.1015	0.2382	0.3615	0.4691	0.5321	0.5735
SC _GMM	0.1003	0.2322	0.3655	0.4745	0.5344	0.5705

Table 10
Experiment on PCA projected, random projected and raw ANN-SIFT-1M in terms of mAP for ABAH_IM.

# Bits	16	32	64	128	256	512
PCA	**0.0254**	**0.0926**	**0.2129**	**0.3471**	**0.4630**	**0.5403**
Random	0.0100	0.0360	0.1116	0.2739	0.4261	0.5400
Raw	0.0083	0.0379	0.1332	0.2953	0.4417	0.5063

Table 11
Experiment on PCA projected, random projected and raw ANN-SIFT-1M in terms of Recall@1000 for ABAH_IM.

# Bits	16	32	64	128	256	512
PCA	**0.1104**	**0.2434**	**0.3750**	**0.4835**	**0.5608**	**0.6116**
Random	0.0570	0.1236	0.2505	0.4221	0.5484	0.6037
Raw	0.0534	0.1353	0.2866	0.4441	0.5591	0.5862

From the experiment results, we can see that our methods perform better than IsoH in terms of both recall and mAP. Since we adaptively allocate different numbers of bits to encode dimensions with different dispersions, the important dimensions in the original space will be still important in Hamming space. Hence, adaptive bit allocation scheme may be a better alternative to dealing with anisotropy of the projected data.

6. Conclusion

In this work, we propose a new hashing method to embed the real-value vectors into a neighborhood structure-preserved Hamming space for ANN search. The dispersions of different dimensions after projection are not the same, and it is unreasonable to use the same number of bits to encode them. We adopt the adaptive bit allocation scheme to adaptively use different numbers of bits to encode different dimensions. For those dimensions that have a larger dispersion, we use more bits to encode them and fewer bits for the dimensions with smaller dispersion. This can preserve the original neighborhood structure. Our methods are very simple and intuitive. Extensive experiments have demonstrated the effectiveness of our method. The proposed method can be combined with some other projection techniques like kernel PCA. Besides, combined with distance metric learning techniques, our ABAH methods may capture the semantic structures of the data. In future, we will tackle these issues.

Acknowledgments

This work has been supported by the National Key Technology R&D Program of China under Grant nos. 2012BAH04F02, 2012BAH88F02, 2013BAH61F01 and 2013BAH63F01 and the International S&T Cooperation Program of China under Grant no. 2013DFG12980.

Appendix A. Supplementary materials

Supplementary data associated with this article can be found in the online version at http://dx.doi.org/10.1016/j.neucom.2014.10.042.

References

[1] D. Lowe, Distinctive image features from scale invariant keypoints, Int. J. Comput. Vis. 66 (2) (2004) 91–110.
[2] Jon Louis Bentley, *K–d* trees for semidynamic point sets, in: Proceedings of the Symposium on Computational Geometry, 1990, pp. 187–197.
[3] P. Li, J. Cheng, H. Lu, Hashing with dual complementary projection learning for fast image retrieval, Neurocomputing 120 (2013) 83–89.
[4] A. Gionis, P. Indyk, R. Motwani, Similarity search in high dimensions via hashing, in: Proceedings of International Conference on Very Large Data Base, 1999, pp. 518–529.
[5] J. He, W. Liu, S.F. Chang, Scalable similarity search with optimized kernel hashing, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2010, pp. 1129–1138.
[6] W. Liu, J. Wang, Y. Mu, S. Kumar, S.F. Chang, Compact hyperplane hashing with bilinear functions, in: Proceedings of International Conference on Machine Learning, 2012.
[7] B. Stein, Principles of hash-based text retrieval, in: Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval, 2007, pp. 527–534.
[8] C. Yao, J. Bu, C. Wu, G. Chen, Semi-supervised spectral hashing for fast similarity search, Neurocomputing 101 (2013) 52–58.
[9] D. Zhang, J. Wang, D. Cai, J. Lu, Self-taught hashing for fast similarity search, in: Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval, 2010, pp. 18–25.
[10] Y. Zhen, D.Y. Yeung, A probabilistic model for multimodal hash function learning, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2012, pp. 940–948.
[11] S. Arya, D. Mount, N. Netanyahu, R. Silverman, A. Wu, An optimal algorithm for approximate nearest neighbor searching fixed dimensions, J. ACM 45 (6) (1998) 891–923.
[12] T. Liu, A. Moore, A. Gay, K. Yang, An investigation of practical approximate nearest neighbor algorithm, in: Proceedings of Conference on Advances in Neural Information Processing Systems, 2004, pp. 32–33.
[13] M. Datar, N. Immorlica, P. Indyk V. Mirrokni, Locality-sensitive hashing scheme based on p-stable distributions, in: Proceedings of the 20th Annual Symposium on Computational Geometry, 2004, pp. 253–262.
[14] Y. Weiss, A. Torralba, R. Fergus, Spectral hashing, in: Proceedings of Conference on Advances in Neural Information Processing Systems, 2008, pp. 1753–1760.
[15] M. Raginsky, S. Lazebnik, Locality-sensitive binary codes from shift-invariant kernels, in: Proceedings of Conference on Advances in Neural Information Processing Systems, 2009, pp. 1509–1517.
[16] G. Hinton, R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (5786) (2006) 504–507.
[17] G. Shakhnarovich, P.A. Viola, T. Darrell, Fast pose estimation with parameter-sensitive hashing, in: Proceedings of the International Conference on Computer Vision, 2003, pp. 750–757.
[18] Y. Zhuang, Y. Liu, F. Wu, Y. Zhang, J. Shao, Hypergraph spectral hashing for similarity search of social image, in: Proceedings of ACM Conference on Multimedia, 2011, pp. 1457–1460.
[19] Y. Zhen, D.Y. Yeung, Active hashing and its application to image and text retrieval, Data Min. Knowl. Discov. 26 (2) (2013) 255–274.
[20] H. Xu, J. Wang, Z. Li, G. Zeng, S. Li, N. Yu, Complementary hashing for approximate nearest neighbor search, in: Proceedings of the International Conference on Computer Vision, 2011, pp. 1631–1638.
[21] J. Wang, S. Kumar, S.F. Chang, Semi-supervised hashing for scalable image retrieval, in: Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition, 2010, pp. 3424–3431.
[22] J. He, R. Radhakrishnan, S.F. Chang, C. Bauer, Compact hashing with joint optimization of search accuracy and time, in: Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition, 2011, pp. 753–760.
[23] B. Kulis, T. Darrell, Learning to hash with binary reconstruction embeddings, in: Proceedings of Conference on Advances in Neural Information Processing Systems, 2009, pp. 1042–1050.
[24] M. Norouzi, D.J. Fleet, Minimal loss hashing for compact binary codes, in: Proceedings of the International Conference on Machine Learning, 2011, pp. 353–360.
[25] C. Strecha, A. Bronstein, M. Bronstein, P. Fua, Ldahash: improved matching with smaller descriptors, IEEE Trans. Pattern Anal. Mach. Intell. 34 (1) (2012) 66–78.
[26] H. Jégou, M. Douze, C. Schmid, Hamming embedding and weak geometric consistency for large scale image search, in: Proceedings of the European Conference on Computer Vision, 2008, pp. 304–317.
[27] B. Wang, Z. Li, M. Li, Efficient duplicate image detection algorithm for web images and large-scale database, Technical report, Microsoft Research, 2005.
[28] W. Liu, J. Wang, S. Kumar, S. Chang, Hashing with graphs, in: Proceedings of the International Conference on Machine Learning, 2011, pp. 1–8.
[29] W. Kong, W.J. Li, Isotropic hashing, in: Proceedings of Conference on Advances in Neural Information Processing Systems, 2012, pp. 1655–1663.
[30] H. Jégou, M. Douze, C. Schmid, Product quantization for nearest neighbor search, IEEE Trans. Pattern Anal. Mach. Intell. 33 (1) (2011) 117–127.
[31] M.J. Huiskes, M.S. Lew, The MIR Flickr retrieval evaluation, in: Proceedings of the ACM International Conference on Multimedia Information Retrieval, 2008, pp. 39–43.
[32] D. Nistér H. Stewénius, Scalable recognition with a vocabulary tree, in: Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition, 2006, pp. 2161–2168.
[33] A. Krizhevsky, Learning Multiple Layers of Features from Tiny Images (MS. thesis), 2009.
[34] A. Torralba, R. Fergus, W. Freeman, 80 million tiny images: a large data set for nonparametric object and scene recognition, IEEE Trans. Pattern Anal. Mach. Intell. 30 (11) (2008) 1958–1970.
[35] A. Oliva, A. Torralba, Modeling the shape of the scene: a holistic representation of the spatial envelop, Int. J. Comput. Vis. 42 (3) (2001) 145–175.
[36] R.S. Lin, D. Ross, J. Yagnik, Spec hashing: similarity preserving algorithm for entropy-based coding, in: Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition, 2010, pp. 848–854.
[37] Q.Z. Guo, Z. Zeng, S. Zhang, Y. Zhang, F. Wang, Adaptive bit allocation hashing for approximate nearest neighbor search, in: Proceedings of the IEEE International Conference on Multimedia and Expo, 2013, pp. 1–6.
[38] Y. Gong, S. Lazebnik, Iterative quantization: a procrustean approach to learning binary codes, in: Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition, 2011, pp. 817–824.
[39] M.M. Esmaeili, R.K. Ward, M. Fatourechi, A fast approximate nearest neighbor search algorithm in the Hamming space, IEEE Trans. Pattern Anal. Mach. Intell. 34 (12) (2012) 2481–2488.
[40] J. Lin, E. Keogh, L. Wei, S. Lonardi, Experiencing SAX: a novel symbolic representation of time series, Data Min. Knowl. Discov. 15 (2) (2007) 107–144.
[41] Anand D. Subramaniam, Bhaskar D. Rao, Pdf optimized parametric vector quantization of speech line spectral frequencies, IEEE Trans. Speech Audio Process. 11 (2) (2003) 130–142.
[42] Jonas Lindblom, Jonas Samuelsson, Bound support Gaussian mixture modeling of speech spectra, IEEE Trans. Speech Audio Process. 11 (1) (2003) 88–99.
[43] Zhanyu Ma, Arne Leijon, W. Bastiaan Kleijn, Vector quantization of LSF parameters with a mixture of Dirichlet distributions, IEEE Trans. Audio Speech Lang. Process. 21 (9) (2013) 1777–1790.

Qin-Zhen Guo received the B.S. degree in Automation from Hunan University in 2011. He is currently pursuing the Ph. D. degree at the High-tech Innovation Center, Institute of Automation, Chinese Academy of Sciences, Beijing, China. His research interests include image retrieval, machine learning, and pattern recognition.

Zhi Zeng received the B.S. and M.S. degrees in Computer Science in 2003 and 2006, respectively, both from Chongqing University, China, and the Ph.D. degree in Pattern Recognition from the Institute of Automation, Chinese Academy of Sciences, in 2009. He is currently a Senior Engineer in the Institute of Automation, Chinese Academy of Sciences. His research interests include information retrieval, machine learning, multimedia, and digital rights management.

Shuwu Zhang got his Ph.D. from Chinese Academy of Sciences in 1997. Currently, he is a professor of Institute of Automation, Chinese Academy of Sciences. His research interests are focused on digital content analysis, digital right management, and web-based cultural content service technologies.