ADAPTIVE BIT ALLOCATION HASHING FOR APPROXIMATE NEAREST NEIGHBOR SEARCH

Qin-Zhen Guo, Zhi Zeng, Shuwu Zhang, Yuan Zhang, Fangyuan Wang

Institute of Automation, Chinese Academy of Sciences
{qinzhen.guo, zhi.zeng, shuwu.zhang, yuan.zhang, fangyuan.wang}@ia.ac.cn

ABSTRACT

Using hashing algorithms to learn binary codes representation of data for fast approximate nearest neighbor (ANN) search has attracted more and more attentions. 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 of these methods is that every dimension of the projected data is regarded as the same importance and represented by one bit, which may result in ineffective codes. We introduce an adaptive bit allocation hashing (ABAH) method to encode data for ANN search. The basic idea is, according to the dispersion of every dimension after projection we use different number of bits to encode them. ABAH can effectively preserve the neighborhood structure in the original data space. Extensive experiments show that ABAH significantly outperforms three state-of-the-art methods.

Index Terms— Approximate nearest neighbor search, hamming embedding, adaptive bit allocation, image retrieval.

1. INTRODUCTION

Nearest neighbor (NN) search has been widely used in computer vision application, like scene classification, 3D reconstruction and other related application areas. 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 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 neighbors and several approximate nearest neighbor (ANN) search techniques have been developed including tree-based methods and hash-based methods. Since the tree-based methods [2] are turned out to be not more efficient than exhaustive search for high dimensions, hash-based ANN techniques which aim at embedding the data into Hamming space have attracted more and more attentions. 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, i.e. the codes should be locality-sensitive. Searching 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 hashing methods is that the Hamming distance between two codes can be efficiently computed by XOR operator. Moreover, the storage will be largely reduced for storing the binary codes. Therefore, considering the fast query speed and low storage cost, hashing has been a popular candidate for efficient ANN search in large scale datasets.

To generate a *k*-bit binary code, the hashing methods usually need *k* 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. The pioneering work locality-sensitive hashing (LSH) [3] employs simple random projections for the first step. Spectral hashing (SH) [4], 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, Restricted Boltzmann Machine (RBM) [5, 6] and Boosting Similarity Sensitive Coding (BoostSSC) [7]. LDAHash [8] adopts Linear Discriminant Analysis (LDA) to learn the projection matrix and chooses the threshold by optimizing a loss function. Iterative quantization (ITQ) [9] introduces a procrustean approach that minimizes quantization loss to learn the projection matrix. In Hamming Embedding (HE) [10], Jégou et al. randomly draws a matrix of Gaussian values and then apply a QR factorization to it. The projection matrix is formed by the first rows of the orthogonal matrix obtained by this decomposition. Finally, use median value to binarize every dimension. Similarly, in order to computes *k*-bit hash codes, PCA Hashing [11]

projects data to the k principal components, and then use average value to binarize the coefficients. Most hashing methods regard every dimension of vectors as the same importance and allocate one bit to represent every dimension. Different with this, Anchor Graph Hashing (AGH) [12] uses a two-layer hash functions to quantize every dimension. But in AGH, every dimension is still encoded by the same number of bits.

In this paper, we propose an adaptive bit allocation hashing (ABAH) approach to adaptively allocating different number of bits to encode every dimension for ANN search. The neighborhood structure in the original space can be better preserved in the Hamming space. Extensive experiments demonstrate the superiority of our method. The rest of the paper is organized as follows. In Section 2, we describe the details of our ABAH method. Experimental results are presented in Section 3. The paper is concluded in Section 4.

2. ADAPTIVE BIT ALLOCATION HASHING

This section describes the details of our ABAH method. First, we introduce the motivation of ABAH. Then, the ABAH scheme is proposed. After that, the whole learning procedure for ABAH will be summarized. Finally, we do some discussions about ABAH.

2.1. Motivation

Given $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)$ and $\boldsymbol{y} = (y_1, y_2, \ldots, y_d)$, which are two randomly chosen d-dim data, the Euclidean distance between $\boldsymbol{x}$ and $\boldsymbol{y}$, $D(\boldsymbol{x}, \boldsymbol{y}) = \mathrm{sqrt}((x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_d - y_d)^2)$ is mainly determined by those dimensions that have a larger value of $(x_i - y_i)^2$. Hence, if the data on some dimensions have a larger $E((x_i - y_i)^2)$, which is related to the dispersion of the ith dimension, those dimensions will be more important for computing Euclidean distance. When embedding the data into Hamming space, in order to maintain the neighborhood structure, important dimensions in the Euclidean space should be also important in the Hamming space. To achieve this goal, we adaptively allocate bits to encode dimensions according their dispersion. If we use the same, fewer bits to encode every dimension, the information loss of dimensions with larger dispersion caused by encoding will increase. But using the same, more bits to encode all the dimensions will be superfluous for dimensions with less dispersion.

As illustrated in Figure 1, the projection values of the data on direction w_1 have a larger dispersion and are dominant for Euclidean distance computation. More bits will be allocated to encode that dimension (after projection) for preserving neighborhood structure.

2.2. ABAH Scheme

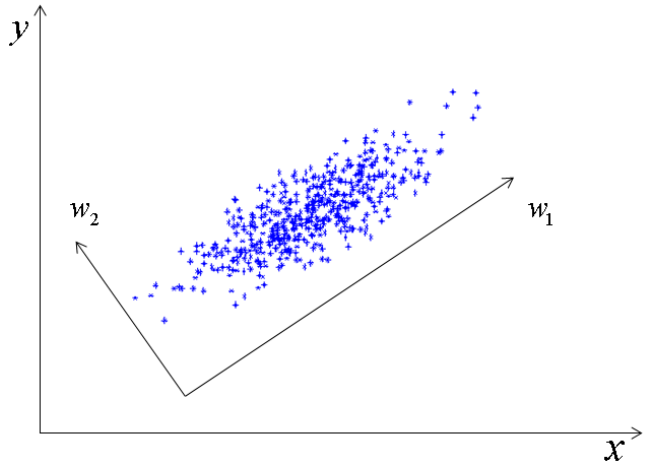


Figure 1. Illustration of different dimensions' effect on Euclidean distance. Obviously, after projection, the values on direction w_1 have a larger dispersion and are dominant for Euclidean distance computation.

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, dimension's dispersion is calculated. Then according to the dispersion of every dimension, different number of bits are allocated to encode corresponding dimension.

2.2.1. 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 k 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 2.2.2) and can be easily used to compute every component's code length (see Section 2.2.3).

It's important to note that all the data discussed below are PCA-projected data.

2.2.2. Dispersion Measurement

As proved below, variance is equivalent to $E((x_i - y_i)^2)$ and better to measure dispersion for maintaining neighborhood structure under the assumption that the training data are uncorrelated.

First of all, we introduce some notations. We have a set of N data $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$, $\boldsymbol{x}_i \in \mathbb{R}^d$, that form the rows of the data matrix $\boldsymbol{X} \in \mathbb{R}^{N \times d}$.

For the pth column of $\boldsymbol{X}$,

$$\begin{aligned} E((x_{ip} - x_{jp})^2) &= E(x_{ip}^2 + x_{jp}^2 - 2x_{ip}x_{jp}) \\ &= 2Ex_{ip}^2 - 2Ex_{ip}x_{jp} \end{aligned} \tag{1}$$

Assume the training data are independent and identically distributed (i.i.d.), and then we have,

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

The variance of the pth column of $\boldsymbol{X}$ is,

$$\begin{aligned} Var_p &= E((x_{ip} - \mu)^2) \\ &= E(x_{ip}^2 + \mu^2 - 2x_{ip}\mu) \\ &= Ex_{ip}^2 + \mu^2 - 2\mu Ex_{ip} \\ &= Ex_{ip}^2 - \mu^2 \end{aligned} \quad (3)$$

From the derivation, we can see that variance is equivalent to $E((x_i - y_i)^2)$. μ is the mean of the pth column of $\boldsymbol{X}$.

Besides, the variance of every dimension is easy to obtain 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.

2.2.3. Calculating Code Length

To maintain the neighborhood structure, we use more bits to encode those dimensions having larger variance. In other words, if the eigenvalue corresponding to the ith column of the PCA projection matrix is larger, we will use more bits to encode the vector's ith dimension.

Simply and intuitively, given total code length k, the ith dimension's code length will be

$$c_i = \left[k \cdot \frac{\lambda_i}{\sum_{n=1}^{d} \lambda_n} + 0.5 \right] \quad (4)$$

λ_i is the eigenvalue corresponding to the ith dimension of the PCA projection matrix and [x] means floor(x). One problem of (4) is that if someλ_i are very close to each other and they are not large enough, e.g., $\lambda_i < 0.5 \cdot (1/k) \sum_{n=1}^{d} \lambda_n$, this will result in those $c_i = 0$ and $\sum_{i=1}^{d} c_i \neq k$. To handle this problem, we use a variant strategy as follows,

$$c_i = \begin{cases} \left[k \cdot \frac{\lambda_i}{\sum_{n=i}^{d} \lambda_n} + 0.5 \right] & i = 1 \\ \left[(k - \sum_{t=1}^{i-1} c_t) \cdot \frac{\lambda_i}{\sum_{n=i}^{d} \lambda_n} + 0.5 \right] & i \geq 2 \end{cases} \quad (5)$$

Then, every dimension is encoded sequentially. Since PCA will find the principal components with degressive variance, the codes length of every dimension have a downward trend. If $c_i = 0$ according to (5), and $\sum_{t=1}^{i-1} c_t \neq k$, we set $c_i = 1$.

The proposed method in Eq. 5 is not an optimal solution. For example, if the first i-1 components use less than k bits, the ith component which may not be large enough to use one bit will be assigned one. In the future, we will tackle this.

2.2.4. 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 $\boldsymbol{x}_p = (x_{p1}, x_{p2}, \ldots, x_{pd})$, c_i bits are allocated to encode the ith dimension. Firstly, we divide the range of the ith dimension into c_i+1 parts by c_i thresholds. The c_i thresholds are chosen in a simple way,

$$t_j^i = x_i^{\min} + (j / (c_i + 1)) \cdot (x_i^{\max} - x_i^{\min}) \quad (6)$$

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

This simple algorithm has one limitation. The thresholds are chosen uniformly from the range of every dimension. This may divide two values close to each other into two different regions. We have experimented with k-means method to find out every dimension's "thresholds". Specifically speaking, for the ith dimension we use 1-dim k-means algorithm to find (c_i+1) centroids with ascending order. Values belong to the ith cluster will be encoded by subcode s_i, which consists of (c_i-i+1) zeros followed by (i-1) ones. The k-means algorithm can give a better partition and the centroids are better representation of every parts. It achieves better results and we name it ABAH_KM.

2.3. Summary of ABAH Learning

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

- Finding the principal components of the original data using PCA.
- Calculating the code length of every dimension in the way described in Section 2.2.3.

- According to every dimension's thresholds, using different number of bits to encode them in the way described in Section 2.2.4.
- Concatenating the subcodes to obtain the final binary code.

2.4. Discussion

Now let us discuss the time complexity of ABAH. The training time is mainly determined by the step of encoding every dimension (see Section 2.2.4). Once we get the codes 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 in constant time by looking up the table. To find the thresholds, in the worst case, the time consuming is O(kN). k is the total bits number and N is the number of training samples. Since our methods can generate codes longer than data dimensions, if k is larger than data dimension d, the time consuming of computing thresholds is at most O(dN). For one dimension, it only needs O(N) time to compute the thresholds. 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(kN).

When testing, there are two strategies. One is that we linearly search every code in the database to find the nearest neighbor. The other is using hash table or inverted list to organize the database, which can achieve sub-linear search time. For simplicity, we adopt the first method and the time complexity is O(kN).

3. EXPERIMENT

In this section we evaluate and compare our two methods with three state-of-the-art methods.

3.1. Datasets

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

- ANN-GIST-150K: A set of 960 dimensional, 150K GIST descriptors, which consist of a subset of ANN-GIST-1M [13].
- ANN-SIFT-150K: A set of 128 dimensional, 150K SIFT descriptors, which consist of a subset of ANN-SIFT-1M [13].
- ANN-GIST-1M: A set of 960 dimensional, 1M GIST descriptors.
- ANN-SIFT-1M: A set of 128 dimensional, 1M SIFT descriptors.

For datasets ANN-GIST-150K, ANN-SIFT-150K, 100K vectors randomly chosen are used as training set and the rest 50K vectors are used as test set. For ANN-GIST-1M and ANN-SIFT-1M, we randomly choose 200K vectors as queries. The rest 800K vectors are used as training set. The ground truth is defined by k nearest neighbors computed by the exhaustive, linear scan based on the Euclidean distance. The performance is measured by mean Average Precision (mAP).

Finally, we employ our methods for practical application for image retrieval on the dataset UKB.

- UKB: The University of Kentucky Benchmark (UKB) [14] contains 10,200 images of 2,550 objects. 4 pictures correspond to each object, taken from different angles.

We represent each image with a 512 (4×8×16) dimensional GIST descriptor [15]. 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×precision@4. For learning purpose, we use an independent image set to learn the parameters (codes length and thresholds of every dimension). One important thing we should know is that the retrieval results not only depend on the hashing method, but also are affected by the image representation to a great extent.

For all the experiments, we use a machine consisting of i7-2600 CPU and 16GB main memory.

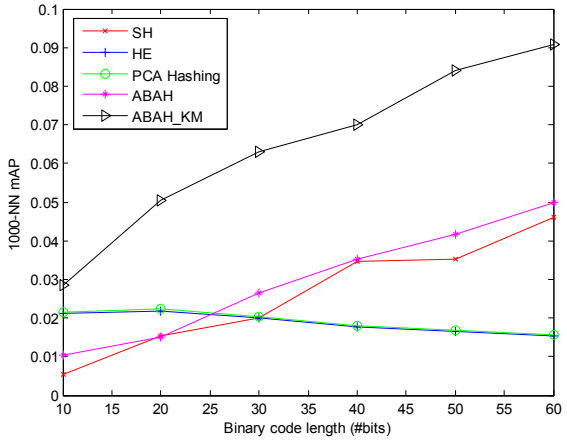
3.2. Compared Methods

- SH: Spectral Hashing [4]
- PCA Hashing: PCA Hashing [11] uses the average value of every dimension to binarize this dimension with one bit after PCA projection.
- HE: Hamming Embedding (HE) [10]. For simplicity, we use PCA projection matrix. Different from PCA Hashing, HE uses the median value of every dimension as the threshold.
- ABAH: The hashing method is proposed in this paper with the thresholds obtained uniformly.
- ABAH_KM: An improved version of our ABAH with the thresholds obtained by k-means algorithm.

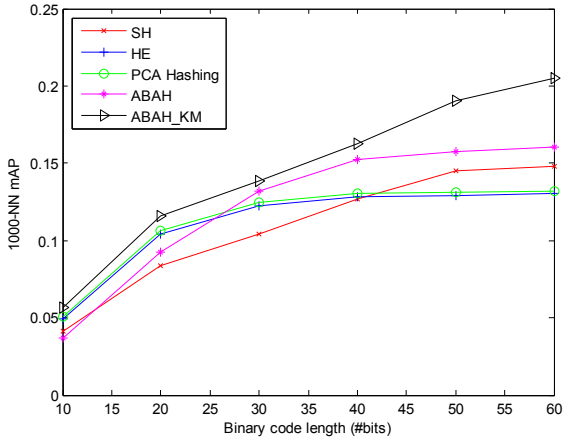
3.3. Results and Analysis

Figure 2 shows the mAP of k nearest search of all the tested methods for k = 1000 on ANN-GIST-150K, ANN-SIFT-150K. Our ABAH_KM method performs better over all the tested methods across all the tested bit lengths ranging from 10 bits to 60 bits. Similar to that reported in [16], PCA Hashing performs worse when using more bits encode the data. And so is HE. Although HE uses median value to binarize every dimension while PCA Hashing uses average value, they almost have the same results (actually, PCA Hashing performs slightly better than HE).

As we can see, ABAH_KM achieves constantly better results than ABAH on the two datasets from 10 bits to 60

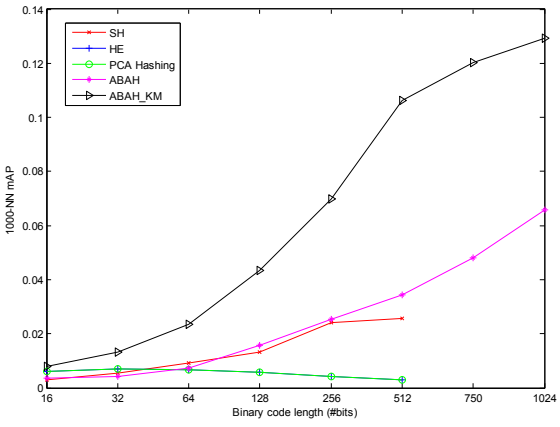


(a)



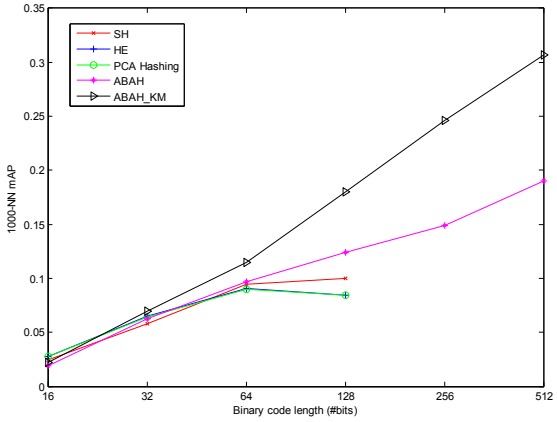
(b)

Figure 2. Comparison of ABAH to state-of-the-art methods: (a) experiment on ANN-GIST-150K dataset. (b) experiment on ANN-SIFT-150K dataset.



(a)

bits, especially when using fewer bits to encode the data. This is mainly because the cluster centers got by *k*-means



(b)

Figure 3. Comparison of ABAH to state-of-the-art methods: (a) experiment on ANN-GIST-1M dataset. (b) experiment on ANN-SIFT-1M dataset.

algorithm are better representations of the data and can give a better partition.

One phenomenon worthy our attention is that a 10-bit code can't represent one vector uniquely. Because 10-bit codes at most have 1024 different representations while we have far more vectors in the dataset. Our ABAH_KM method also gets significantly better results in this case.

In Figure 3, experimental results on dataset ANN-GIST-1M and ANN-SIFT-1M are presented. For dataset ANN-GIST-1M, PCA projection is used to project the data to 512 dimensions. Since SH, HE, PCA Hashing can't generate codes longer than data dimension while our ABAH and ABAH_KM can, we don't plot the results when codes are longer than 512 bits for SH, HE, PCA Hashing. By the same token, we don't plot the results of 256-bit and 512-bit for SH, HE, PCA Hashing in Fig. 3(b). We can see that our ABAH_KM method achieves the best results most of time. With codes longer than data dimension for ABAH and ABAH_KM, better results are obtained. One can strike a balance between accuracy and efficiency by choosing appropriate number of bits. Experimental results about precision and recall on dataset ANN-GIST-1M can be seen in the supplemental material.

Figure 4 shows the search performance in terms of 1000-NN mAP as a function of the size of the dataset (up to 1M). All the compared methods use 40 bits to encode the vectors. One can observe that our ABAH_KM approach achieves the best results on different size of dataset. We also experiment on two small datasets, ANN-GIST-15K and ANN-GIST-60K. The results are presented in the supplemental material.

Finally, for image retrieval task, we test our methods on dataset UKB.

As shown in Figure 5. Our ABAH_KM hashing approach outperforms other compared methods in all ranges

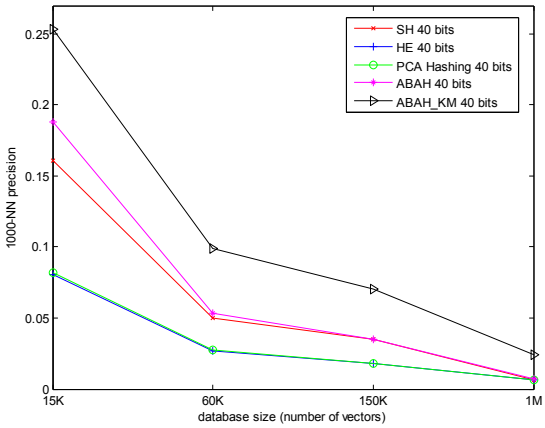


Figure 4. Comparison of ABAH to state-of-the-art methods. 1000-NN mAP for ANN-GIST-1M as function of the size of the dataset.

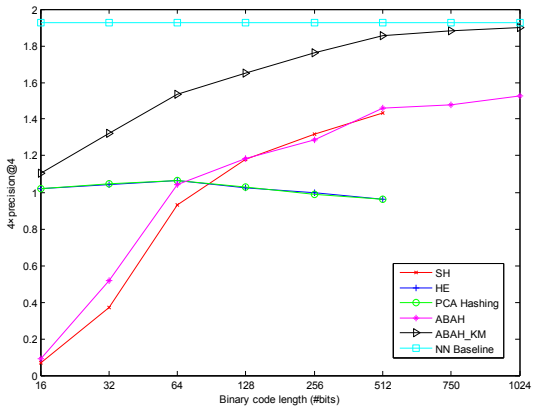


Figure 5. Comparison of ABAH to state-of-the-art methods on UKB dataset.

from 16 bits to 512 bits. NN Baseline is computed by the exhaustive, linear scan based on the Euclidean distance. HE and PCA Hashing achieve surprisingly good results with shorter codes in this dataset. Since images are represented by 512-bit dimensional vectors, same to the case of ANN-GIST-1M, we don't plot the results of 750-bit and 1024-bit for SH, HE, PCA Hashing. We can see that our ABAH_KM can better approximate the results of linear search.

4. 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. It is very simple and intuitive. Extensive experiments have demonstrated the effectiveness of our method. The proposed method can be combined with some other projection technique like kernel PCA, LDA. In the future, we will tackle these issues.

5. ACKNOWLEDGMENT

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

6. REFERENCES

[1] D. Lowe, "Distinctive image features from scale-invariant keypoints," *IJCV*, vol. 66, pp. 91-110, 2004.

[2] Jon Louis Bentley, "K-d trees for semidynamic point sets," in *Symposium on Computational Geometry*, pp. 187-197, 1990.

[3] M. Datar, N. Immorlica, P. Indyk and V. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Symposium on Computational Geometry*, pp. 153-262, 2004.

[4] Y. Weiss, A. Torralba, and R. Fergus, "Spectral Hashing," in *NIPS*, pp. 1753-1760, 2008.

[5] G. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, 18(7): 1527-1554, 2006.

[6] G. Hinton, and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, 313(5786): 504-507, 2006.

[7] G. Shakhnarovich, P. A. Viola, and T. Darrell, "Fast pose estimation with parameter-sensitive hashing," in *ICCV*, pp. 750-759, 2003.

[8] C. Strecha, A. Bronstein, M. Bronstein, and P. Fua, "Ldahash: Improved matching with smaller descriptors," *IEEE TPAMI*, vol. 34, no. 1, pp. 66-78, 2012.

[9] Y. Gong and S. Lazebnik, "Iterative quantization: a procrustean approach to learning binary codes," in *CVPR*, 2011.

[10] H. Jégou, M. Douze, and C. Schmid, "Hamming embedding and weak geometric consistency for large scale image search," in *ECCV*, Oct. 2008.

[11] B. Wang, Z. Li, and M. Li, "Efficient duplicate image detection algorithm for web images and large-scale database," Technical report, Microsoft Research, 2005.

[12] W. Liu, J. Wang, S. Kumar, and S. Chang, "Hashing with graphs," in *ICML*, 2011.

[13] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE TPAMI*, vol. 33, no. 1, pp. 117-127, 2011.

[14] D. Nistér and H. Stewénius, "Scalable recognition with a vocabulary tree," in *CVPR*, pp. 2161-2168, 2006.

[15] A. Oliva and A. Torralba, "Modeling the shape of the scene: a holistic representation of the spatial envelop," *IJCV*, 2001.

[16] R.-S. Lin, D. Ross, and J. Yagnik, "Spec hashing: Similarity preserving algorithm for entropy-based coding," in *CVPR*, pp. 848-854, 2010.