Manhattan Hashing for Large-Scale Image Retrieval

Weihao Kong
Shanghai Key Laboratory of Scalable Computing and Systems
Department of Computer Science and Engineering
Shanghai Jiao Tong University
Shanghai, China
kongweihao@cs.sjtu.edu.cn

Wu-Jun Li
Shanghai Key Laboratory of Scalable Computing and Systems
Department of Computer Science and Engineering
Shanghai Jiao Tong University
Shanghai, China
liwujun@cs.sjtu.edu.cn

Minyi Guo
Shanghai Key Laboratory of Scalable Computing and Systems
Department of Computer Science and Engineering
Shanghai Jiao Tong University
Shanghai, China
guo-my@cs.sjtu.edu.cn

ABSTRACT

Hashing is used to learn binary-code representation for data with expectation of preserving the neighborhood structure in the original feature space. Due to its fast query speed and reduced storage cost, hashing has been widely used for efficient nearest neighbor search in a large variety of applications like text and image retrieval. Most existing hashing methods adopt Hamming distance to measure the similarity (neighborhood) between points in the hashcode space. However, one problem with Hamming distance is that it may destroy the neighborhood structure in the original feature space, which violates the essential goal of hashing. In this paper, Manhattan hashing (MH), which is based on Manhattan distance, is proposed to solve the problem of Hamming distance based hashing. The basic idea of MH is to encode each projected dimension with multiple bits of *natural binary code* (NBC), based on which the Manhattan distance between points in the hashcode space is calculated for nearest neighbor search. MH can effectively preserve the neighborhood structure in the data to achieve the goal of hashing. To the best of our knowledge, this is the first work to adopt Manhattan distance with NBC for hashing. Experiments on several large-scale image data sets containing up to one million points show that our MH method can significantly outperform other state-of-the-art methods.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Search and Retrieval

General Terms

Algorithms, Measurement

Keywords

Hashing, Image Retrieval, Approximate Nearest Neighbor Search, Hamming Distance, Manhattan Distance

SIGIR'12, August 12–16, 2012, Portland, Oregon, USA.

1. INTRODUCTION

Nearest neighbor (NN) search [28] has been widely used in machine learning and related application areas, such as information retrieval, data mining, and computer vision. Recently, with the explosive growth of data on the Internet, there has been increasing interest in NN search in massive (large-scale) data sets. Traditional brute force NN search requires scanning all the points in a data set whose time complexity is linear to the sample size. Hence, it is computationally prohibitive to adopt brute force NN search for massive data sets which might contain millions or even billions of points. Another challenge faced by NN search in massive data sets is the excessive storage cost which is typically unacceptable if traditional data formats are used.

To solve these problems, researchers have proposed to use hashing techniques for efficient approximate nearest neighbor (ANN) search [1, 5, 7, 19, 30, 38, 39, 41]. The goal of hashing is to learn binary-code representation for data which can preserve the neighborhood (similarity) structure in the original feature space. More specifically, each data point will be encoded as a compact binary string in the hashcode space, and similar points in the original feature space should be mapped to close points in the hashcode space. By using hashing codes, we can achieve constant or sub-linear search time complexity [32]. Moreover, the storage needed to store the binary codes will be dramatically reduced. For example, if each point is represented by a vector of 1024 bytes in the original space, a data set of 1 million points will cost 1GB memory. On the contrary, if we hash each point into a vector of 128 bits, the memory needed to store the data set of 1 million points will be reduced to 16MB. Therefore, hashing provides a very effective way to achieve fast query speed with low storage cost, which makes it a popular candidate for efficient ANN search in massive data sets [1].

To avoid the NP-hard solution which directly computes the best binary codes for a given data set [36], most existing hashing methods adopt a learning strategy containing two stages: projection stage and quantization stage. In the projection stage, several projected dimensions of real values are generated. In the quantization stage, the real values generated from the projection stage are quantized into binary codes by thresholding. For example, the widely used single-bit quantization (SBQ) strategy adopts one single bit to quantize each projected dimension. More specifically, given a point $\mathbf{x}$ from the original space, each projected dimension i will be associated with a real-valued projection function $f_i(\mathbf{x})$. The ith hash bit of $\mathbf{x}$ will be 1 if $f_i(\mathbf{x}) \geq \theta$. Otherwise, it will be 0. Here, θ is a threshold, which is typically set to 0 if the data have been normalized to have zero mean. Although a lot of projection methods have been proposed for hashing, there exist only two quantization

methods. One is the SBQ method stated above, and the other is the hierarchical quantization (HQ) method in anchor graph hashing (AGH) [18]. Rather than using one bit, HQ divides each dimension into four regions with three thresholds and uses two bits to encode each region. Hence, HQ will associate each projected dimension with two bits. Figure 1 (a) and Figure 1 (b) illustrate the results of SBQ and HQ for one projected dimension, respectively. Till now, only one hashing method, AGH in [18], adopts HQ for quantization. All the other hashing methods adopt SBQ for quantization.

Currently, almost all hashing methods adopt Hamming distance to measure the similarity (neighborhood) between points in the hashcode space. The Hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different [1]. As will be stated below in Section 3.1, neither SBQ nor HQ can effectively preserve the neighborhood structure under the constraint of Hamming distance. Hence, although the projection functions in the projection stage can preserve the neighborhood structure, the whole hashing procedure will still destroy the neighborhood structure in the original feature space due to the limitation of Hamming distance. This will violate the goal of hashing and consequently satisfactory performance cannot be easily achieved by traditional Hamming distance based hashing methods.

In this paper, we propose to use Manhattan distance for hashing to solve the problem of existing hashing methods. The result is our novel hashing method called Manhattan hashing (MH). The main contributions of this paper are briefly outlined as follows:

- Although a lot of hashing methods have been proposed, most of them focus on the projection stage while ignoring the quantization stage. This work will systematically study the effect of quantization. We find that the quantization stage is at least as important as the projection stage. A good quantization strategy combined with a bad projection strategy may achieve better performance than a bad quantization strategy combined with a good projection strategy. This finding is very interesting, which might stimulate other researchers to move their attention from the projection stage to the quantization stage, and finally propose better methods simultaneously taking both stages into consideration.

- MH encodes each projected dimension with multiple bits of *natural binary code* (NBC), based on which the Manhattan distance between points in the hashcode space is calculated for nearest neighbor search. MH can effectively preserve the neighborhood structure in the data to achieve the goal of hashing. To the best of our knowledge, this is the first work to adopt Manhattan distance with NBC for hashing.

- Experiments on several large-scale image data sets containing up to one million points show that our MH method can significantly outperform other state-of-the-art methods.

The rest of this paper is organized as follows. In Section 2, we introduce the related work of our method. Section 3 describes the details of our MH method. Experimental results are presented in Section 4. Finally, we conclude the whole paper in Section 5.

2. RELATED WORK

Due to the promising performance in terms of either speed or storage, hashing has been widely used for efficient ANN search in a large variety of applications with massive data sets, such as text retrieval [33, 39], image retrieval [6, 20], audio retrieval [2], and near-duplicate video retrieval [29]. As a consequence, many hashing methods have been proposed by researchers. In general, the existing methods can be roughly divided into two main classes [6, 39]: data-independent methods and data-dependent methods [2].

The representative data-independent methods include locality-sensitive hashing (LSH) [1, 5] and its extensions [3, 16, 17, 21, 24]. The hash functions of these methods are just some simple random projections which are independent of the training data. Shift invariant kernel hashing (SIKH) [24] adopts projection functions which are similar to those of LSH, but SIKH applies a shifted cosine function to generate hash values. Many applications, such as image retrieval [24] and cross-language information retrieval [38], have adopted these data-independent hashing methods for ANN. Generally, data-independent methods need longer codes than data-dependent methods to achieve satisfactory performance [6]. Longer codes means higher storage and computational cost. Hence, the data-independent methods are less efficient than the data-dependent methods.

Recently, data-dependent methods, which try to learn the hash functions from the training data, have attracted more and more attentions by researchers. Semantic hashing [25, 26] adopts a deep generative model based on restricted Boltzmann machine (RBM) [9] to learn the hash functions. Experiments on text retrieval demonstrate that semantic hashing can achieve better performance than the original TF-IDF representation [27] and LSH. AdaBoost [4] is adopted by [2] to learn hash functions from weakly labeled positive samples. The resulting hashing method achieves better performance than LSH for audio retrieval. Spectral hashing (SH) [36] uses spectral graph partitioning strategy for hash function learning where the graph is constructed based on the similarity between data points. To learn the hash functions, binary reconstruction embedding (BRE) [15] explicitly minimizes the reconstruction error between the distances in the original feature space and the Hamming distances of the corresponding binary codes. Semi-supervised hashing (SSH) [34, 35] exploits both labeled data and unlabeled data for hash function learning. Self-taught hashing [39] uses some self-labeled data to facilitate the supervised hash function learning. Complementary hashing [37] exploits multiple complementary hash tables learned sequentially in a boosting manner to effectively balance the precision and recall. Composite hashing [38] combines multiple information sources into the hash function learning procedure. Minimal loss hashing (MLH) [22] tries to formulate the hashing problem as a structured prediction problem based on the latent structural SVM framework. SPICA [8] tries to find independent projections by jointly optimizing both accuracy and time. Hypergraph hashing [42] extends SH to hypergraph to model the high-order relationships between social images. Active hashing [40] is proposed to actively select the most informative labels for hash function learning. Iterative quantization (ITQ) [6] tries to learn an orthogonal rotation matrix to refine the initial projection matrix learned by principal component analysis (PCA) [13]. Experimental results show that ITQ can achieve better performance than most state-of-the-art methods.

Few of the existing methods discussed above have studied the effect of quantization. Because existing quantization strategies can not effectively preserve the neighborhood structure under the constraint of Hamming distance, most existing hashing methods still can not achieve satisfactory performance even though a large number of sophisticated projection functions have been designed by

[1] http://en.wikipedia.org/wiki/Hamming_distance

[2] In [39], data-independent is called data-oblivious while data-dependent is called data-aware. It is obvious that they have the same meaning.

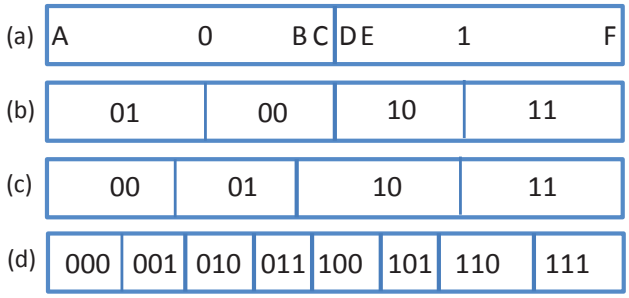
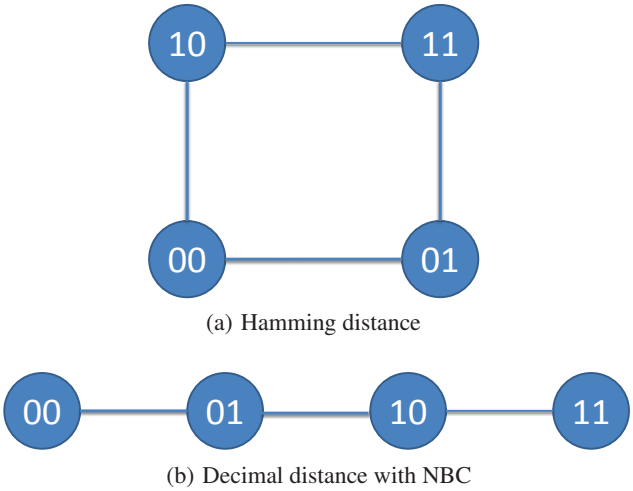


Figure 1: Different quantization methods: (a) single-bit quantization (SBQ); (b) hierarchical quantization (HQ); (c) 2-bit Manhattan quantization (2-MQ); (d) 3-bit Manhattan quantization (3-MQ).



(a) Hamming distance

(b) Decimal distance with NBC

Figure 2: Hamming distance and Decimal distance between 2-bit codes. The distance between two points (i.e., nodes in the graph) is the length of the shortest path between them.

researchers. The work in this paper tries to study these important factors which have been ignored by existing works.

3. MANHATTAN HASHING

This section describes the details of our Manhattan hashing (MH) method. First, we will introduce the motivation of MH. Then, the Manhattan distance driven quantization strategy will be proposed. After that, the whole learning procedure for MH will be summarized. Finally, we will do some qualitative analysis about the performance of MH.

3.1 Motivation

Given a point $\mathbf{x}$ from the original feature space $\mathbb{R}^d$, hashing tries to encode it with a binary string of c bits via the mapping $h : \mathbb{R}^d \rightarrow \{0, 1\}^c$. As said in Section 1, most hashing methods adopt a two-stage strategy to learn h because directly learning h is an NP-hard problem. Let's first take SBQ based hashing as an example for illustration. In the projection stage, c real-valued projection functions $\{f_k(\mathbf{x})\}_{k=1}^c$ are learned and each function can generate one real value. Hence, we have c *projected dimensions* each of which corresponds to one projection function. In the quantization stage, the real-values are quantized into a binary string by thresholding. More specifically, $h_k(\mathbf{x}) = 1$ if $f_k(\mathbf{x}) \geq \theta$. Otherwise, $h_k(\mathbf{x}) = 0$. Here, we assume $h(\mathbf{x}) = [h_1(\mathbf{x}), h_2(\mathbf{x}), \cdots, h_c(\mathbf{x})]^T$, and θ is a threshold which is typically set to 0 if the data have been normalized to have zero mean. Figure 1 (a) illustrates the result of SBQ for one projected dimension. Currently, most hashing methods adopt SBQ for the quantization stage. Hence, the difference between these methods lies in the different projection functions.

Till now, only two quantization methods have been proposed for hashing. One is SBQ just discussed above, and the other is HQ which is adopted by only one hashing method AGH [18]. Rather than using one bit, HQ divides each projected dimension into four regions with three thresholds and uses two bits to encode each region. Hence, to get a c-bit code, HQ based hashing need only $c/2$ projection functions. Figure 1 (b) illustrates the result of HQ for one projected dimension.

To achieve satisfactory performance for ANN, one important requirement of hashing is to preserve the neighborhood structure in the original space. More specifically, close points in the original space $\mathbb{R}^d$ should be mapped to similar binary codes in the code space $\{0, 1\}^c$.

We can easily find that with Hamming distance, both SBQ and HQ will destroy the neighborhood structure in the data. As illustrated in Figure 1 (a), point 'C' and point 'D' will be quantized into 0 and 1 respectively although they are very close to each other in the real-valued space. On the contrary, point 'D' and point 'F' will be quantized into the same code 1 although they are far away from each other. Hence, in the code space of this dimension, the Hamming distance between 'F' and 'D' is smaller than that between 'C' and 'D', which obviously indicates that SBQ can destroy the neighborhood structure in the original space.

HQ can also destroy the neighborhood structure of data. Let $d_h(x, y)$ denote the Hamming distance between binary codes x and y. From Figure 1 (b), we can get $d_h(A, F) = d_h(A, B) = d_h(C, D) = d_h(D, F) = 1$, and $d_h(A, D) = d_h(C, F) = 2$. Hence, we can find that the Hamming distance between the two farthest points 'A' and 'F' is the same as that between two relatively close points such as 'A' and 'B'. The even worse case is that $d_h(A, F) < d_h(A, D)$, which is obviously very unreasonable.

The problem of HQ is inevitable under the constraint of Hamming distance. Figure 2 (a) shows the Hamming distance between different 2-bit codes, where the distance between two points (i.e., nodes in the graph) is equivalent to the length of the shortest path between them. We can see that the largest Hamming distance between 2-bit codes is 2. However, to keep the relative distances between 4 different points (or regions), the largest distance between two different 2-bit codes should be at least 3. Hence, no matter how we permute the 2-bit codes for the four regions in Figure 1 (b), we cannot get any neighborhood-preserving result under the constraint of Hamming distance. One choice to overcome this problem of HQ is to design a new distance measurement.

3.2 Manhattan Distance Driven Quantization

As stated above, the problem that HQ cannot preserve the neighborhood structure in the data is essentially from the Hamming distance. Here, we will show that Manhattan distance with natural binary code (NBC) can solve the problem of HQ.

The Manhattan distance between two points is the sum of the differences on their dimensions. Let $\mathbf{x} = [x_1, x_2, \cdots, x_d]^T$, $\mathbf{y} = [y_1, y_2, \cdots, y_d]^T$, the Manhattan distance between $\mathbf{x}$ and $\mathbf{y}$ is de-

fined as follows:

$$d_m(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{d} |x_i - y_i|, \quad (1)$$

where $|x|$ denotes the absolute value of x.

To adapt Manhattan distance for hashing, we adopt a q-bit quantization scheme. More specifically, after we have learned the real-valued projection functions, we divide each projected dimension into 2^q regions and then use q bits of *natural binary code* (NBC) to encode the index of each region. For example, if $q = 2$, each projected dimension is divided into 4 regions, and the indices of these regions are $\{0, 1, 2, 3\}$, the NBC codes of which are $\{00, 01, 10, 11\}$. If $q = 3$, the indices of regions are $\{0, 1, 2, 3, 4, 5, 6, 7\}$, and the NBC codes are $\{000, 001, 010, 011, 100, 101, 110, 111\}$. Figure 1 (c) shows the quantization result with $q = 2$ and Figure 1 (d) shows the quantization result with $q = 3$. Because this quantization scheme is driven by Manhattan distance, we call it *Manhattan quantization* (MQ). The MQ with q bits is denoted as q-MQ.

Another issue for MQ is about threshold learning. Badly learned thresholds will deteriorate the quantization performance. To achieve the neighborhood-preserving goal, we need to make the points in each region as similar as possible. In this paper, we use k-means clustering algorithm [14] to learn the thresholds from the training data. More specifically, if we need to quantize each projected dimension into q bits, we use k-means to cluster the real values of each projected dimension into 2^q clusters, and the midpoint of the line joining neighboring cluster centers will be used as thresholds.

In our MH, we use the *decimal distance* rather than the Hamming distance to measure the distances between the q-bit codes for each projected dimension. The decimal distance is defined to be the difference between the decimal values of the corresponding NBC codes. For example, let $d_d(\mathbf{x}, \mathbf{y})$ denote the decimal distance between $\mathbf{x}$ and $\mathbf{y}$, then $d_d(10, 00) = |2 - 0| = 2$ and $d_d(010, 110) = |2 - 6| = 4$. Figure 2 (b) shows the decimal distances between different 2-bit codes, where the distance between two points (i.e., nodes in the graph) is equivalent to the length of the shortest path between them. We can see that the largest decimal distance between 2-bit codes is 3, which is enough to effectively preserve the relative distances between 4 different points (or regions). Figure 1 (c) shows one of the encoding results which can preserve the relative distances between the regions. Figure 1 (d) is the results with $q = 3$. It is obvious that the relative distances between the regions are also preserved. In fact, it is not hard to prove that this nice property will be satisfied for any positive integer q. Hence, our MQ strategy with $q \geq 2$ provides a more effective way to preserve the neighborhood structure than SBQ and HQ.

Given two binary codes $\mathbf{x}$ and $\mathbf{y}$ generated by MH, the Manhattan distance between them is computed from (1), where x_i and y_i correspond to the ith projected dimension which should contain q bits. Furthermore, the difference between two q-bit codes of each dimension should be measured with decimal distance. For example, if $q = 2$,

$$\begin{aligned} d_m(000100, 110000) &= d_d(00, 11) + d_d(01, 00) + d_d(00, 00) \\ &= 3 + 1 + 0 \\ &= 4. \end{aligned}$$

If $q = 3$,

$$\begin{aligned} d_m(000100, 110000) &= d_d(000, 110) + d_d(100, 000) \\ &= 6 + 4 \\ &= 10. \end{aligned}$$

It is easy to see that when $q = 1$, the results computed with Manhattan distance are equivalent to those with Hamming distance, and consequently our MH method degenerates to the traditional SBQ-based hashing methods.

3.3 Summary of MH Learning

Given a training set, the whole learning procedure of MH, including both projection and quantization stages, can be summarized as follows:

- Choose a positive integer q, which is 2 in default;
- Choose an existing projection method or design a new projection method, and then learn $\lfloor \frac{c}{q} \rfloor$ projection functions;
- Use k-means to learn 2^q clusters, and compute $2^q - 1$ thresholds based on the centers of the learned clusters;
- Use MQ in Section 3.2 to quantize each projected dimension into q bits of NBC code based on the learned thresholds;
- Concatenate the q-bit codes of all the $\lfloor \frac{c}{q} \rfloor$ projected dimensions into one long code to represent each point.

One important property of our MH learning procedure is that MH can choose an existing projection method for the projection stage, which means that the novel part of MH is mainly from the quantization stage which has been ignored by most existing hashing methods. By combining different projection functions with our MQ strategy, we can get different versions of MH. For example, if PCA is used for projection, we can get 'PCA-MQ'. If the random projection functions in LSH are used for projection, we can get 'LSH-MQ'. Both PCA-MQ and LSH-MQ can be seen as variants of MH. Similarly, we can design other versions of MH.

3.4 Discussion

Because the MQ for MH can better preserve the neighborhood structure between points, it is expected that with the same projection functions, MH will generally outperform SBQ or HQ based hashing methods. This will be verified by our experiments in Section 4.

The total training time contains two parts: one part is for projection, and the other is for quantization. Compared with SBQ, although MQ need extra $O(n)$ time for k-means learning, the total time complexity of MH is still the same as that of SBQ based methods because the projection time is at least $O(n)$. Here n is the number of training points. Similarly, we can prove that with the same projection functions, MH has the same time complexity as HQ based methods.

It is not easy to compare the absolute training time between MH and traditional SBQ based methods. Although extra training time is needed for MH to perform k-means learning, the number of projection functions will be decreased to $\lfloor \frac{c}{q} \rfloor$ while SBQ based methods need c projection functions. Hence, whether MH is faster or not depends on the specific projection functions. If the projection stage is very time-consuming, MH might be faster than SBQ based methods. But for other cases, SBQ based methods can be faster than MH. The absolute training time of HQ based methods is about the same as that of MH with $q = 2$ because HQ also need to learn the thresholds for quantization. For $q \geq 3$, whether MH is faster than HQ base methods or not depends on the specific projection functions because MH need fewer projection functions but larger number of thresholds.

As for query procedure, the speed of computing hashcode for query in MH is expected to be faster than SBQ based methods because the number of projection operations for MH with $q \geq 2$ is

only $\lfloor \frac{c}{q} \rfloor$ of that for SBQ. The query speed of MH with $q = 2$ is the same as that of HQ based methods. When $q \geq 3$, the speed of computing hashcode of MH will be faster than HQ based methods due to the smaller number of projection operations.

4. EXPERIMENT

4.1 Data Sets

To evaluate the effectiveness of our MH method, we use three publicly available image sets, LabelMe [32][3], TinyImage [31][4], and ANN_SIFT1M [12] [5].

The first data set is *22K LabelMe* used in [22, 32]. LabelMe is a web-based tool designed to facilitate image annotation. With the help of this annotation tool, the current LabelMe data set contains as large as 200,790 images which span a wide variety of object categories. Most images in LabelMe contain multiple objects. *22K LabelMe* contains 22,019 images sampled from the large LabelMe data set. As in [32], we scale the images to have the size of 32x32 pixels, and represent each image with 512-dimensional GIST descriptors [23].

The second data set is *100K TinyImage* containing 100,000 images randomly sampled from the original 80 million Tiny Images [31]. TinyImage data set aims to present a visualization of all the nouns in the English language arranged by semantic meaning. A total number of 79,302,017 images were collected by Google's image search engine and other search engines. The original images have the size of 32x32 pixels. As in [31], we represent them with 384-dimensional GIST descriptors [23].

The third data set is *ANN_SIFT1M* introduced in [10, 11, 12]. It consists of 1,000,000 images each represented as 128-dimensional SIFT descriptors. ANN_SIFT1M contains three vector subsets: subset for learning, subset for database, and subset for query. The learning subset is retrieved from Flickr images and the database and query subsets are from the INRIA Holidays images [11]. We conduct our experiments only on the database subset, which consists of 1,000,000 images each represented as 128-dimensional SIFT descriptors.

Figure 3 shows some representative images sampled from LabelMe and TinyImage data sets. Please note that the authors of *ANN_SIFT1M* provide only the extracted features without any original images of their data. From Figure 3, it is easy to see that LabelMe and TinyImage have different characteristics. The LabelMe data set contains high-resolution photos, in fact most of which are street view photos. On the contrary, the images in TinyImage data set have low-resolution.

4.2 Baselines

As stated in Section 3.3, MQ can be combined with different projection functions to get different variants of MH. In this paper, the most representative methods, ITQ [6], SIKH [24], LSH [1], SH [36], and PCA [6, 13], are chosen to evaluate the effectiveness of our MQ strategy. ITQ, SH, and PCA are data-dependent methods, while SIKH and LSH are data-independent methods. These chosen methods are briefly introduced as follows:

- **ITQ:** ITQ uses an iteration method to find an orthogonal rotation matrix to refine the initial projection matrix learned by PCA so that the quantization error of mapping the data to the vertices of binary hypercube is minimized. Experimental results in [6] show that it can achieve better performance than most state-of-the-art methods. In our experiment, we set the iteration number of ITQ to be 100.
- **SIKH:** SIKH uses random projections to approximate the shift-invariant kernels. As in [6, 24], we use a Gaussian kernel whose bandwidth is set to the average distance to the 50th nearest neighbor.
- **LSH:** LSH uses a Gaussian random matrix to perform random projection.
- **SH:** SH uses the eigenfunctions computed from the data similarity graph for projection.
- **PCA:** PCA uses the eigenvectors corresponding to the largest eigenvalues of the covariance matrix for projection.

All the above hashing methods can be used to provide projection functions. By adopting different quantization strategies, we can get different variants of a specific hashing method. Let's take PCA as an example. 'PCA-SBQ' denotes the original PCA hashing method with single-bit quantization, 'PCA-HQ' denotes the combination of PCA projection with HQ quantization [18], and 'PCA-MQ' denotes one variant of MH combining the PCA projection with Manhattan quantization (MQ). Because the threshold optimization techniques for HQ in [18] can not be used for the above five methods, we use the same thresholds as those in MQ. All experiments are conducted on our workstation with Intel(R) Xeon(R) CPU X7560@2.27GHz and 64G memory.

4.3 Evaluation Metrics

We adopt the scheme widely used in recent papers [6, 24, 36] to evaluate our method. More specifically, we define the ground truth to be the Euclidean neighbors in the original feature space. The average distance to the 50th nearest neighbors is used as a threshold to find whether a point is a true positive or not. All the experimental results are averaged over 10 random training/test partitions. For each partition, we randomly select 1000 points as queries, and leave the rest as training set to learn the hash functions.

Based on the Euclidean ground truth, we can compute the precision, recall and the mean average precision (mAP) [6, 18] which are defined as follows:

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

$$Recall = \frac{\text{the number of retrieved relevent points}}{\text{the number of all relevent points}},$$

$$mAP = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{n_i}\sum_{k=1}^{n_i} Precision(R_{ik}),$$

where $q_i \in Q$ is a query, n_i is the number of points relevant to q_i in the data set, the relevant points are ordered as $\{x_1, x_2, \cdots, x_{n_i}\}$, R_{ik} is the set of ranked retrieval results from the top result until you get to point x_k.

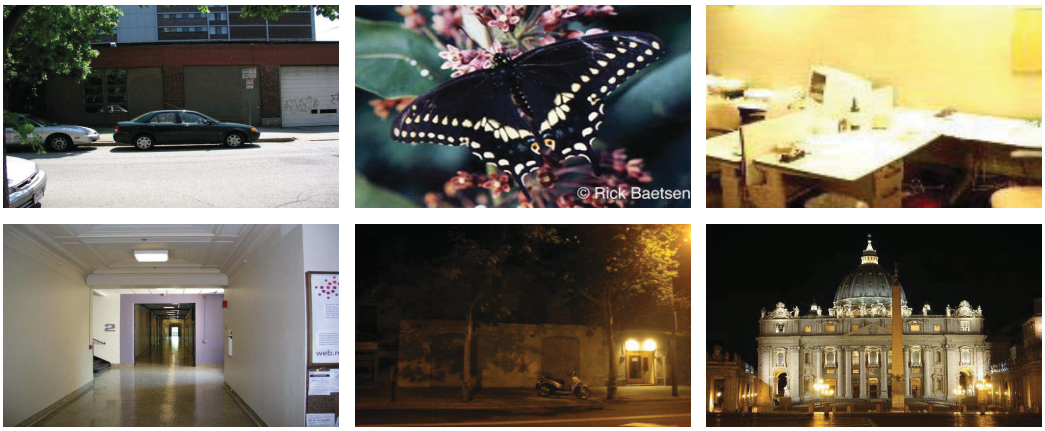
4.4 Results

The mAP values for different methods with different code sizes on *22K LabelMe*, *100K TinyImage*, and ANN_SIFT1M are shown in Table 1, Table 2, and Table 3, respectively. The value of each entry in the tables is the mAP of a combination of a projection function with a quantization method under a specific code size. The best mAP among SBQ, HQ and MQ under the same setting

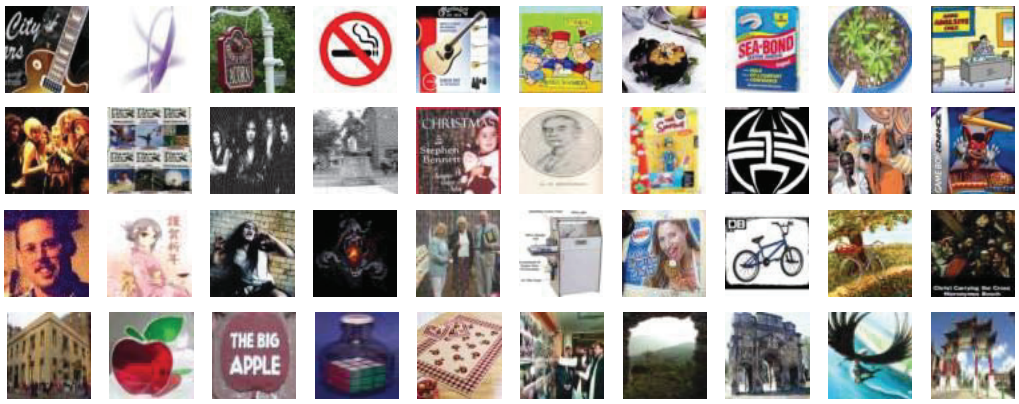
[3] http://labelme.csail.mit.edu/

[4] http://groups.csail.mit.edu/vision/TinyImages/.

[5] http://corpus-texmex.irisa.fr/.



(a) LabelMe



(b) TinyImage

Figure 3: Sample images from LabelMe and TinyImage data sets.

is shown in bold face. To study the effect of q which is the length of NBC for each projected dimension, we evaluate our MH methods on *22K LabelMe* and *100K TinyImage* by setting the q to three different values (2, 3, and 4). From Table 1 and Table 2, we find that $q = 2$ (2-MQ) achieves the best performance for most cases. Hence, unless otherwise stated, $q = 2$ is a default setting.

From Table 1, Table 2, and Table 3, it is easy to find that our MQ method achieves the best performance under most settings, which means that our MQ with Manhattan distance does be very effective. Furthermore, we can also find that HQ achieves better performance than SBQ under most settings. Because both HQ and our MQ methods adopt more than one bit to encode each projected dimension, it may imply that using multiple bits to encode each projected dimension can be better than using just one single bit. This phenomenon has also been observed by the authors of AGH [18]. The essential difference between HQ and 2-MQ lies in the difference between Hamming distance and Manhattan distance. Hence, the better performance of MQ (compared with HQ) shows that Manhattan distance is a better choice (compared with Hamming distance) to preserve the neighborhood (similarity) structure in the data.

Figure 4 and Figure 5 show the precision-recall curves on *22K LabelMe* and ANN_SIFT1M data sets, respectively. Once again, we can easily find that our MQ based MH variants significantly outperform other state-of-the-art methods under most settings.

5. CONCLUSION

Most existing hashing methods focus on the projection stage while ignoring the quantization stage. This work systematically studies the effect of quantization. We find that the quantization stage is at least as important as the projection stage. This work might stimulate other researchers to move their attention from the projection stage to the quantization stage, and finally propose better methods simultaneously taking both stages into consideration.

The existing quantization methods, such as SBQ and HQ, will destroy the neighborhood structure in the original space, which violates the goal of hashing. In this paper, we propose a novel quantization strategy called Manhattan quantization (MQ) to effectively preserve the neighborhood structure among data. The MQ based hashing method, call Manhattan hashing (MH), encodes each projected dimension with multiple bits of *natural binary code* (NBC), based on which the Manhattan distance between points in the hashcode space is calculated for nearest neighbor search. MH can effectively preserve the neighborhood structure in the data to achieve the goal of hashing. To the best of our knowledge, this is the first work to adopt Manhattan distance with NBC for hashing. The effectiveness of our MH method is successfully verified by experiments on several large-scale real-world image data sets.

Table 1: mAP on *22K LabelMe* data set. The best mAP among SBQ, HQ, 2-MQ, 3-MQ and 4-MQ under the same setting is shown in bold face.

# bits	32					64				
	SBQ	HQ	2-MQ	3-MQ	4-MQ	SBQ	HQ	2-MQ	3-MQ	4-MQ
ITQ	0.2771	0.3166	**0.3537**	0.2860	0.2700	0.3283	0.4303	0.4881	**0.5163**	0.4803
SIKH	0.0487	0.0512	**0.0722**	0.0457	0.0339	0.1175	0.1024	**0.1700**	0.1242	0.0906
LSH	**0.1563**	0.1210	0.1382	0.0961	0.0684	0.2577	0.2507	**0.2833**	0.2514	0.1998
SH	0.0802	0.1540	**0.2207**	0.2103	0.2026	0.0988	0.1960	0.3237	0.3440	**0.3441**
PCA	0.0503	0.1414	0.1913	0.2064	**0.2092**	0.0388	0.1585	0.2233	0.3177	**0.3303**
# bits	128					256				
	SBQ	HQ	2-MQ	3-MQ	4-MQ	SBQ	HQ	2-MQ	3-MQ	4-MQ
ITQ	0.3559	0.5203	0.5905	**0.6968**	0.6758	0.3731	0.5862	0.6496	0.8053	**0.8062**
SIKH	0.2673	0.2125	**0.3669**	0.2736	0.2274	0.4109	0.3994	**0.5704**	0.4984	0.4139
LSH	0.3310	0.4360	**0.4596**	0.4423	0.3863	0.3955	0.5768	0.6115	**0.6321**	0.5833
SH	0.1644	0.2553	0.4367	0.4627	**0.4747**	0.2027	0.2671	0.4418	**0.5563**	0.5520
PCA	0.0298	0.1669	0.2114	0.3221	**0.3653**	0.0226	0.1549	0.1710	0.2288	**0.2814**

Table 2: mAP on *100K TinyImage* data set. The best mAP among SBQ, HQ, 2-MQ, 3-MQ and 4-MQ under the same setting is shown in bold face.

# bits	32					64				
	SBQ	HQ	2-MQ	3-MQ	4-MQ	SBQ	HQ	2-MQ	3-MQ	4-MQ
ITQ	**0.4936**	0.3963	0.4279	0.3185	0.3746	0.5811	0.5513	**0.5908**	0.4859	0.5109
SIKH	0.1354	0.1388	**0.2209**	0.1419	0.0868	0.3041	0.2789	**0.4037**	0.3197	0.2023
LSH	**0.3612**	0.3010	0.3396	0.2266	0.2161	0.4843	0.4747	**0.5183**	0.4445	0.4059
SH	0.1173	0.1870	**0.2372**	0.2271	0.1749	0.1934	0.3330	**0.4463**	0.4416	0.3445
PCA	0.0459	0.2124	**0.2569**	0.2566	0.1924	0.0405	0.3316	0.3845	**0.4196**	0.3139
# bits	128					256				
	SBQ	HQ	2-MQ	3-MQ	4-MQ	SBQ	HQ	2-MQ	3-MQ	4-MQ
ITQ	0.6227	0.6882	**0.7346**	0.6853	0.6671	0.6404	0.7877	0.8270	**0.8472**	0.8153
SIKH	0.5077	0.4239	**0.6311**	0.5139	0.3884	0.6625	0.6342	**0.7643**	0.6954	0.5892
LSH	0.5663	0.6421	**0.6856**	0.6474	0.5799	0.6145	0.7678	**0.7855**	0.7854	0.7219
SH	0.3044	0.4610	**0.5664**	0.5620	0.5062	0.4069	0.5643	0.6568	**0.6573**	0.5399
PCA	0.0360	0.4608	0.4896	**0.5137**	0.4533	0.0325	0.5267	0.5348	**0.5382**	0.5157

6. ACKNOWLEDGEMENTS

This work is supported by the NSFC (No. 61100125), the 863 Program of China (No. 2011AA01A202, No. 2012AA011003), and the Program for Changjiang Scholars and Innovative Research Team in University of China (IRT1158, PCSIRT).

7. REFERENCES

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, 2008.

[2] S. Baluja and M. Covell. Learning to hash: forgiving hash functions and applications. *Data Min. Knowl. Discov.*, 17(3):402–430, 2008.

[3] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the ACM Symposium on Computational Geometry*, 2004.

[4] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of International Conference on Machine Learning*, 1996.

[5] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of International Conference on Very Large Data Bases*, 1999.

[6] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *Proceedings of Computer Vision and Pattern Recognition*, 2011.

[7] J. He, W. Liu, and 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.

[8] J. He, R. Radhakrishnan, S.-F. Chang, and C. Bauer. Compact hashing with joint optimization of search accuracy and time. In *Proceedings of Computer Vision and Pattern Recognition*, 2011.

[9] G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.

[10] H. Jegou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *Proceedings of the European Conference on Computer Vision*, 2008.

[11] H. Jegou, M. Douze, and C. Schmid. Improving bag-of-features for large scale image search. *International Journal of Computer Vision*, 87(3):316–336, 2010.

[12] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(1):117–128, 2011.

[13] I. Jolliffe. *Principal Component Analysis*. Springer, 2002.

Table 3: mAP on ANN_SIFT1M data set. The best mAP among SBQ, HQ and 2-MQ under the same setting is shown in bold face.

# bits	32			64			96			128		
	SBQ	HQ	2-MQ	SBQ	HQ	2-MQ	SBQ	HQ	2-MH	SBQ	HQ	2-MQ
ITQ	0.1657	0.2500	**0.2750**	0.4641	0.4745	**0.5087**	0.5424	0.5871	**0.6263**	0.5823	0.6589	**0.6813**
SIKH	0.0394	0.0217	**0.0570**	0.2027	0.0822	**0.2356**	0.2263	0.1664	**0.2768**	0.3936	0.3293	**0.4483**
LSH	0.1163	0.0961	**0.1173**	0.2340	0.2815	**0.3111**	0.3767	0.4541	**0.4599**	0.5329	0.5151	**0.5422**
SH	0.0889	0.2482	**0.2771**	0.1828	0.3841	**0.4576**	0.2236	0.4911	**0.5929**	0.2329	0.5823	**0.6713**
PCA	0.1087	0.2408	**0.2882**	0.1671	0.3956	**0.4683**	0.1625	0.4927	**0.5641**	0.1548	0.5506	**0.6245**

[14] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(7):881–892, 2002.

[15] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *Proceedings of Neural Information Processing Systems*, 2009.

[16] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *Proceedings of International Conference on Computer Vision*, 2009.

[17] B. Kulis, P. Jain, and K. Grauman. Fast similarity search for learned metrics. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(12):2143–2157, 2009.

[18] W. Liu, J. Wang, S. Kumar, and S. Chang. Hashing with graphs. In *Proceedings of International Conference on Machine Learning*, 2011.

[19] W. Liu, J. Wang, Y. Mu, S. Kumar, and S.-F. Chang. Compact hyperplane hashing with bilinear functions. In *Proceedings of International Conference on Machine Learning*, 2012.

[20] Y. Mu, J. Shen, and S. Yan. Weakly-supervised hashing in kernel space. In *Proceedings of Computer Vision and Pattern Recognition*, 2010.

[21] Y. Mu and S. Yan. Non-metric locality-sensitive hashing. In *Proceedings of AAAI Conference on Artificial Intelligence*, 2010.

[22] M. Norouzi and D. J. Fleet. Minimal loss hashing for compact binary codes. In *Proceedings of International Conference on Machine Learning*, 2011.

[23] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42(3):145–175, 2001.

[24] M. Raginsky and S. Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. In *Proceedings of Neural Information Processing Systems*, 2009.

[25] R. Salakhutdinov and G. Hinton. Semantic Hashing. In *SIGIR workshop on Information Retrieval and applications of Graphical Models*, 2007.

[26] R. Salakhutdinov and G. E. Hinton. Semantic hashing. *Int. J. Approx. Reasoning*, 50(7):969–978, 2009.

[27] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5):513–523, 1988.

[28] G. Shakhnarovich, T. Darrell, and P. Indyk. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. The MIT Press, 2006.

[29] J. Song, Y. Yang, Z. Huang, H. T. Shen, and R. Hong. Multiple feature hashing for real-time large scale near-duplicate video retrieval. In *ACM Multimedia*, 2011.

[30] B. Stein. Principles of hash-based text retrieval. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2007.

[31] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(11):1958–1970, 2008.

[32] A. Torralba, R. Fergus, and Y. Weiss. Small codes and large image databases for recognition. In *Proceedings of Computer Vision and Pattern Recognition*, 2008.

[33] F. Ture, T. Elsayed, and J. J. Lin. No free lunch: brute force vs. locality-sensitive hashing for cross-lingual pairwise similarity. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2011.

[34] J. Wang, O. Kumar, and S.-F. Chang. Semi-supervised hashing for scalable image retrieval. In *Proceedings of Computer Vision and Pattern Recognition*, 2010.

[35] J. Wang, S. Kumar, and S.-F. Chang. Sequential projection learning for hashing with compact codes. In *Proceedings of International Conference on Machine Learning*, 2010.

[36] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Proceedings of Neural Information Processing Systems*, 2008.

[37] H. Xu, J. Wang, Z. Li, G. Zeng, S. Li, and N. Yu. Complementary hashing for approximate nearest neighbor search. In *Proceedings of International Conference on Computer Vision*, 2011.

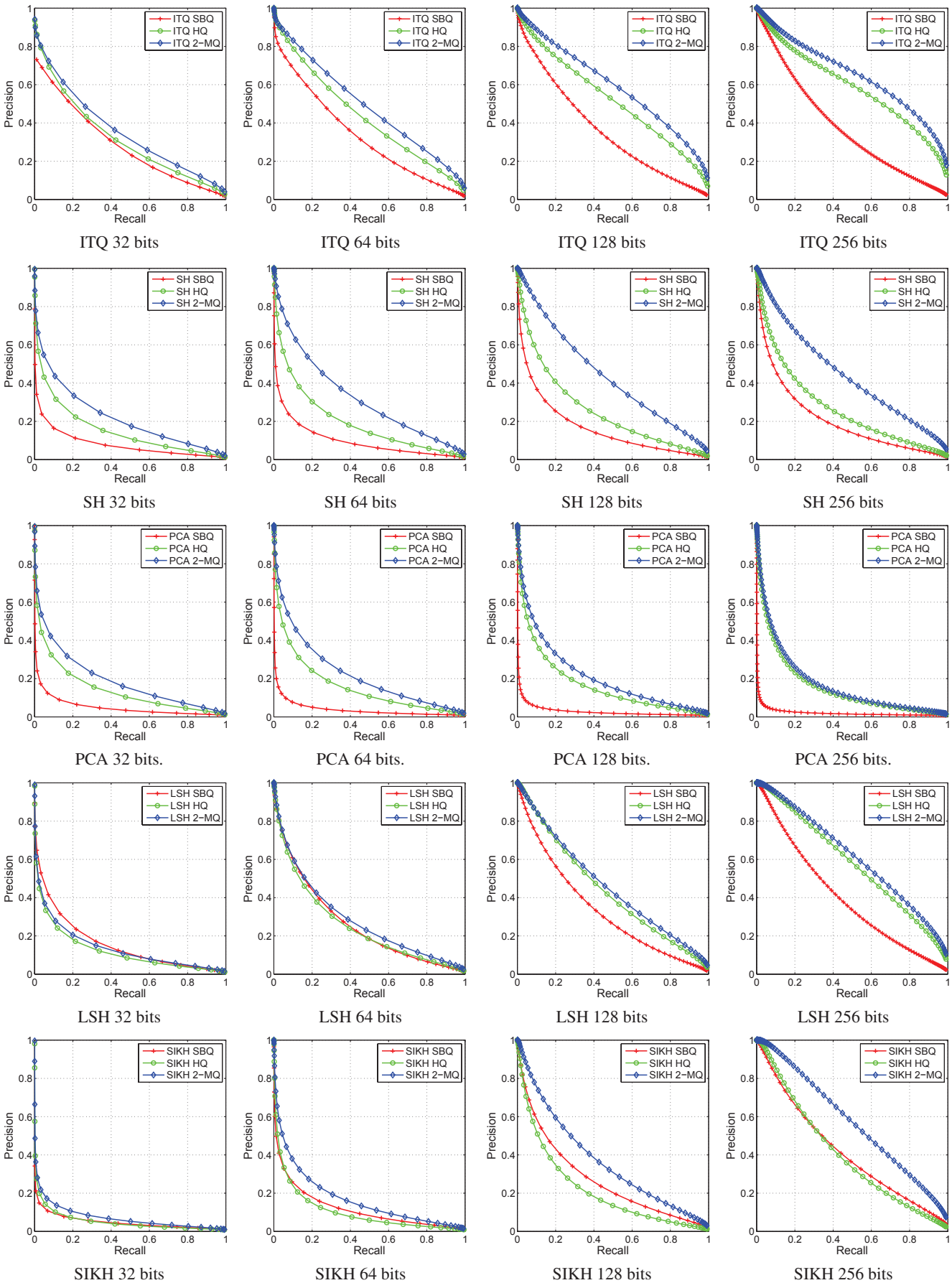
[38] D. Zhang, F. Wang, and L. Si. Composite hashing with multiple information sources. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2011.

[39] D. Zhang, J. Wang, D. Cai, and 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.

[40] Y. Zhen and D.-Y. Yeung. Active hashing and its application to image and text retrieval. *Data Mining and Knowledge Discovery*, 2012.

[41] Y. Zhen and 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.

[42] Y. Zhuang, Y. Liu, F. Wu, Y. Zhang, and J. Shao. Hypergraph spectral hashing for similarity search of social image. In *ACM Multimedia*, 2011.



ITQ 32 bits | ITQ 64 bits | ITQ 128 bits | ITQ 256 bits

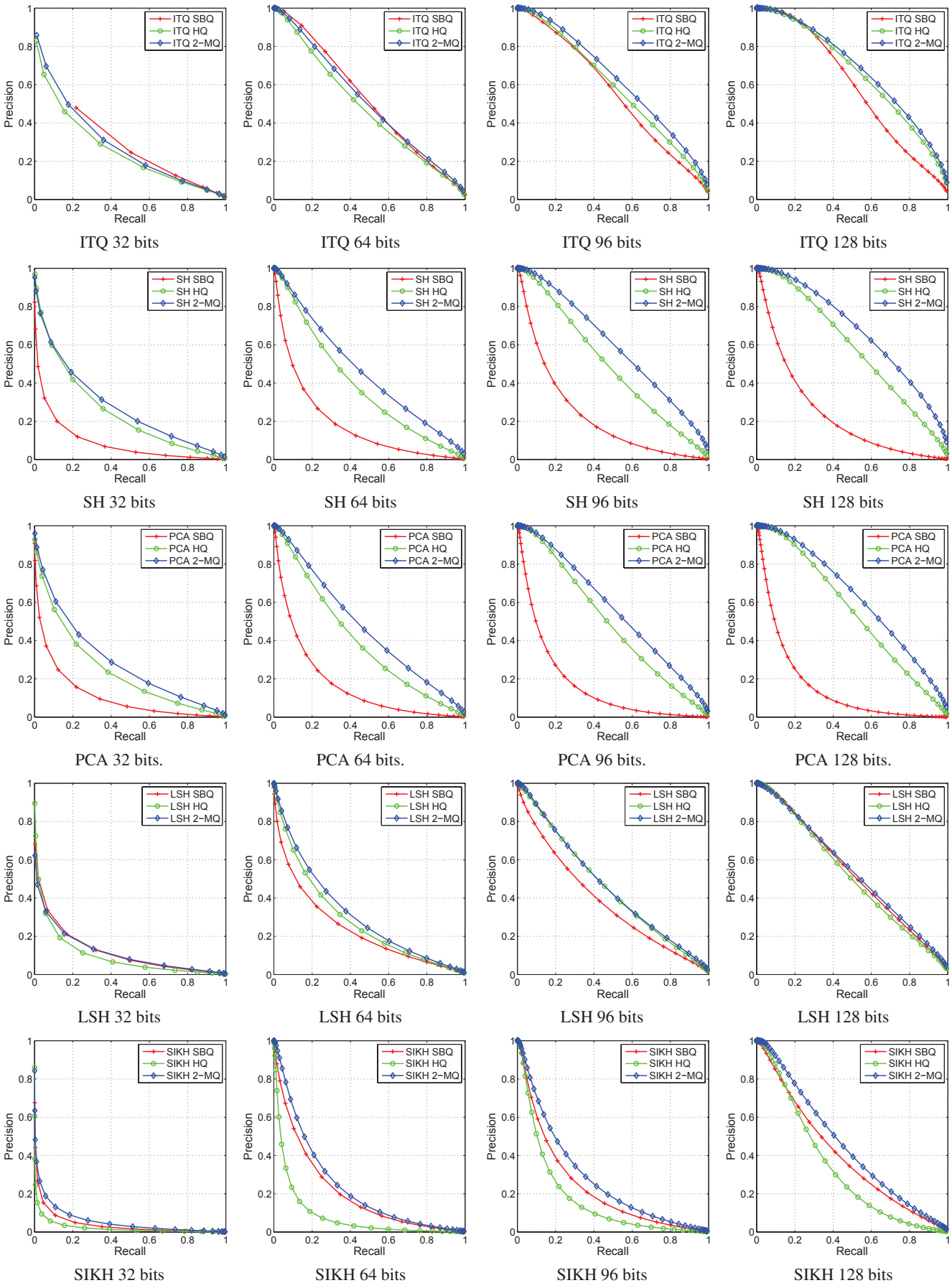
SH 32 bits | SH 64 bits | SH 128 bits | SH 256 bits

PCA 32 bits. | PCA 64 bits. | PCA 128 bits. | PCA 256 bits.

LSH 32 bits | LSH 64 bits | LSH 128 bits | LSH 256 bits

SIKH 32 bits | SIKH 64 bits | SIKH 128 bits | SIKH 256 bits

Figure 4: Precision-recall curve on *22K LabelMe* data set



ITQ 32 bits | ITQ 64 bits | ITQ 96 bits | ITQ 128 bits

SH 32 bits | SH 64 bits | SH 96 bits | SH 128 bits

PCA 32 bits. | PCA 64 bits. | PCA 96 bits. | PCA 128 bits.

LSH 32 bits | LSH 64 bits | LSH 96 bits | LSH 128 bits

SIKH 32 bits | SIKH 64 bits | SIKH 96 bits | SIKH 128 bits

Figure 5: Precision-recall curve on ANN_SIFT1M data set