Binary Multidimensional Scaling for Hashing

Yameng Huang and Zhouchen Lin, *Senior Member, IEEE*

***Abstract*— Hashing is a useful technique for fast nearest neighbor search due to its low storage cost and fast query speed. Unsupervised hashing aims at learning binary hash codes for the original features so that the pairwise distances can be best preserved. While several works have targeted on this task, the results are not satisfactory mainly due to the over-simplified model. In this paper, we propose a unified and concise unsupervised hashing framework, called *binary multidimensional scaling*, which is able to learn the hash code for distance preservation in both batch and online mode. In the batch mode, unlike most existing hashing methods, we do not need to simplify the model by predefining the form of hash map. Instead, we learn the binary codes directly based on the pairwise distances among the normalized original features by alternating minimization. This enables a stronger expressive power of the hash map. In the online mode, we consider the holistic distance relationship between current query example and those we have already learned, rather than only focusing on current data chunk. It is useful when the data come in a streaming fashion. Empirical results show that while being efficient for training, our algorithm outperforms state-of-the-art methods by a large margin in terms of distance preservation, which is practical for real-world applications.**

***Index Terms*— Hashing, approximate nearest neighbor search, large-scale image search, multidimensional scaling.**

I. Introduction

THE recent decade witnessed the flourish of the notion "Big Data". With the explosive growth of data, large-scale information retrieval has attracted significant attention. Given a query, it becomes a challenge to retrieve the desired information from the datasets in the magnitude of millions or billions, because traditional nearest neighbor search that based on linear scan would fail in terms of both efficiency and storage. To overcome this difficulty, efforts have been made to alternative solution, namely *Approximate Nearest Neighbor search* (ANN). This technique has been widely used in document retrieval [11], recommendation system [29], image matching [30], image retrieval [31], object detection [4], etc.

Manuscript received December 7, 2016; revised June 5, 2017, August 3, 2017, and September 15, 2017; accepted September 15, 2017. Date of publication October 4, 2017; date of current version November 3, 2017. The work of Z. Lin was supported in part by the National Basic Research Program of China (973 Program) under Grant 2015CB352502, in part by the National Natural Science Foundation of China under Grant 61625301, Grant 61731018, and Grant 61231002, and in part by Qualcomm. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Dacheng Tao. *(Corresponding author: Zhouchen Lin.)*

The authors are with the Key Laboratory of Machine Perception (Ministry of Education), School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China, and also with the Cooperative Medianet Innovation Center, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: huangyameng@pku.edu.cn; zlin@pku.edu.cn).

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

Digital Object Identifier 10.1109/TIP.2017.2759250

Hashing methods are among the most representative ANN algorithms, which embed high-dimensional real vectors to low-dimensional binary vectors. When additional label information is not available, these algorithms can be further categorized as unsupervised hashing. The goal of unsupervised hashing is to learn a hash map, namely a set of functions. Each function takes the original features as input and generates a binary output. By concatenating this set of outputs, we get the binary codes of the examples, which are supposed to best preserve the similarity (distance) structure [28]. This could be viewed as a multidimensional scaling problem [2] constrained in Hamming space. Unfortunately this problem is NP-hard due to its discrete and non-convex nature. In order to get the results in a reasonable time, we have to resort to its approximation. To obtain a better solution, we may need to minimally relax the constraints in the original problem. The design of such an algorithm is still a great challenge.

Another challenge in real applications is that data sometimes become available in a streaming fashion. In addition, for a truly large scale dataset, data are usually stored on a distributed disk group and are too large to be loaded into memory. We have to turn to online hashing in these scenarios. The main difficulty here is to reasonably model the connection between newly arrived data and the data we have already seen so that the holistic distance relationship between the example pairs could be best preserved.

In this paper, we present a novel scheme for unsupervised hashing, called Binary Multidimensional Scaling (BMDS), which tries to address the challenges of both batch learning methods and online learning methods we have mentioned above. Our goal is to learn the binary codes such that the pairwise distances between example points can be preserved. In the batch mode, we generate the binary codes directly by Alternating Minimization, based on the original features, without predefining the form of hash functions or over-relaxing the original problem. This makes the binary codes inference accurate and scalable. We also have a tailored out-of-sample extension, namely the algorithm to generate the hash codes for the examples that are not in the training database, which is able to preserve the holistic pairwise distances between the new queries and the database. In the online mode, we learn the connection between the current chunk and the previous ones when updating the hash functions, without sacrificing the efficiency. It turns out by experiments that the results are more accurate. In summary, the main contributions of this work are as follows:

- We propose a novel formulation and an optimization method for binary codes inference without predefining the form of hash functions. This enables the maximum expressive power of binary hash codes generation.

The proposed method is efficient and accurate in distance preservation.

- We design a tailored out-of-sample extension which is effective in generating binary codes for new data. We not only validate the performance empirically, but also provide a theoretical analysis on the error between our solution and the globally optimal one.
- Our algorithm also enables an efficient online learning extension, which studies the distance relationship between the current query example and those we have already learned. Thus it outperforms the state-of-the-art online hashing methods in terms of the retrieval accuracy. This is valuable in real-world applications when the data are provided in sequence.

II. Related Work

Representative ANN algorithms could be roughly categorized as tree-based, vector quantization (VQ), and hashing methods. Tree-based methods [7] seek to store the reference examples in a tree, the complexity of which is usually logarithmic time with respect to the number of examples. It is fast for search, but may suffer from memory constraints. VQ [8], [14] methods either build inverted indexing or quantize vectors into codewords for search. Both schemes enable efficient sub-linear time retrieval and light memory cost, while preserving the precision to some degree. Hashing methods map the original features into low-dimensional binary codes such that the neighborhood structure in the original space is preserved. With the binary-code representation, it enables fast nearest neighbor search by using lookup tables or Hamming distance based ranking, which is even faster than VQ. It is also efficient for large-scale data storage. In a word, these methods have quite different design philosophy and achieve different tradeoff in efficiency and accuracy. Thus they have their own advantage and the usage of which relies on the real-world scenarios. In this paper, we will focus on hashing-based methods.

Due to its importance, many efforts have been devoted to the research of hashing algorithms. Based on whether the analysis of a given dataset is required, there are two categories of hashing techniques: data-independent and data-dependent. *Locality-Sensitive Hashing* (LSH) [9] and its variants [19], [29] are representative data-independent methods which generate hash codes via random projections. Although they are ensured to have high collision probability for similar data items, in practice they usually need long hash bits and multiple hash tables to achieve both high precision and recall, which may restrict their applications. So in this paper we mainly discuss data-dependent methods. Subject to the availability of additional label information, these methods can be further classified as unsupervised, supervised, and semi-supervised [34]. Unsupervised methods [10], [12], [17], [23], [24], [26], [38] try to preserve the Euclidean distances (similarities) which are calculated in the original feature space, while in supervised [5], [20], [22], [27] or semi-supervised [33], [39] settings, label-based similarities could be incorporated to narrow the semantic gap. Since additional label information is usually unavailable in real applications, we concentrate on unsupervised hashing in this work. Representative unsupervised hashing methods include *Iterative Quantization* (ITQ) [12], *Circulant Binary Embedding* (CBE) [38], *Bilinear Binary Embeding* (BBE) [10], *Asymmetric Inner-product Binary Coding* (AIBC) [26], *Spectral Hashing* (SH) [36], *Anchor Graph Hashing* (AGH) [23], *Binary Reconstructive Embedding* (BRE) [17], *Discrete Graph Hashing* (DGH) [21], and *Scalable Graph Hashing* (SGH) [15].

As mentioned above, unsupervised hashing is in essence a nonconvex and discrete problem. Besides relaxing the variables to continuous ones, previous works further simplified the problem. For example, ITQ, CBE, BBE, and AIBC further constrained the hash map to be linear perceptrons. SH uses eigen-function as the hash map. AGH, SGH, DGH, and BRE use graph, which relies on the choice of anchor points, to approximate the holistic similarity (distance) relationship. These simplifications usually suffer from the loss of information, thus may lead to unsatisfactory accuracy.

As for online hashing, currently there is relatively few works in this aspect. Among these approaches, *Online Kernel-based Hashing* (OKH) [13] and *Adaptive Hashing* (AdaHash) [3] update the parameters of hash functions according to pairwise similarity of the newly available data pair and meanwhile stay close to the old one. *Online Sketch Hashing* (OSH) [18] learns a small size sketch for each data chunk and updates the hash functions based on the data sketch. While being efficient, these methods update the parameters only based on newly arrived example pair or chunk. That is to say, they do not explicitly build the model based on all the examples that appeared so far. Thus the result might be inaccurate in preserving the holistic distances.

III. Background

Let $\mathbf{X} \in \mathbb{R}^{m\times n}$ denote the training examples in the database, where n is the number of training examples and m is the dimension of features in the original feature space. Each column of $\mathbf{X}$ denotes an example in the database. The goal of our algorithm is to learn the binary codes $\mathbf{Y} \in \{-1, 1\}^{d\times n}$ for training examples as well as $\mathbf{y} \in \{-1, 1\}^d$ for each out-of-sample query $\mathbf{x} \in \mathbb{R}^m$, so that the pairwise distances are well preserved. Different objective functions, such as similarity-distance product minimization (SDPM) [36], similarity-similarity difference minimization (SSDM) [35], and distance-distance difference minimization (DDDM) [17], lead to different algorithms.

Since we want the pairwise distance to be best preserved by our hashing scheme, our algorithm falls into the class of DDDM. The goal of DDDM is to minimize $\sum_{i,j}(d^o_{ij} - d^h_{ij})^2$, so that the difference between the original feature distance d^o_{ij} and the Hamming distance d^h_{ij} is as small as possible. *Binary Reconstructive Embedding* (BRE) [17] belongs to this group. The objective function is:

$$\min \sum_{i,j}\left(\frac{1}{2}\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2 - \frac{1}{2d}\|\boldsymbol{y}_i - \boldsymbol{y}_j\|_2^2\right)^2. \tag{1}$$

It aims at learning parameters $\{w_{mt}\}$ for a set of kernel-based hash functions $\{h_m(\cdot)\}$, such that

$$y_m = h_m(\boldsymbol{x}) = \operatorname{sgn}\left(\sum_{t=1}^{T_m} w_{mt} K(\boldsymbol{s}_{mt}, \boldsymbol{x})\right), \quad m = 1, 2, \cdots, d, \tag{2}$$

where $\{\boldsymbol{s}_{mt}\}$ are anchor points, T_m is the total number of anchor points, and $K(\cdot,\cdot)$ is a kernel function.

One should be reminded that $\|\boldsymbol{y}_i\|_2^2 = d$ is a constant in (1). If we assume that $\boldsymbol{x}_i$ is normalized to have unit ℓ_2 norm,[1] (1) can be transformed to the Maximum Inner Product Search problem (MIPS) [29]:

$$\begin{aligned} \boldsymbol{Y}^* &= \operatorname{argmin}_{\boldsymbol{Y}} \sum_{i,j} \left(\frac{1}{2}\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2 - \frac{1}{2d}\|\boldsymbol{y}_i - \boldsymbol{y}_j\|_2^2\right)^2 \\ &= \operatorname{argmin}_{\boldsymbol{Y}} \sum_{i,j} \|d\boldsymbol{x}_i^T \boldsymbol{x}_j - \boldsymbol{y}_i^T \boldsymbol{y}_j\|^2 \\ &= \operatorname{argmin}_{\boldsymbol{Y}} \|\boldsymbol{Y}^T\boldsymbol{Y} - d\boldsymbol{X}^T\boldsymbol{X}\|_F^2, \quad s.t. \quad \boldsymbol{Y} \in \{-1, 1\}^{d\times n}. \end{aligned} \tag{3}$$

Instead of directly solving (3), Shrivastava and Li [29] propose an "Asymmetric LSH" scheme, which is based on random projection, to generate the binary codes so that the inner products between the examples could be preserved with high probability. Since it is data-independent, the accuracy might be low. *Asymmetric Inner-product Binary Coding* (AIBC) [26] provides an asymmetric data-dependent, namely learning-based solution, to learn the optimal in (3). Specifically, it solves the following problem:

$$\min_{\boldsymbol{W},\boldsymbol{R}} \|\operatorname{sgn}(\boldsymbol{W}^T\boldsymbol{A})^T \operatorname{sgn}(\boldsymbol{R}^T\boldsymbol{X}) - d\boldsymbol{A}^T\boldsymbol{X}\|^2, \tag{4}$$

where $\boldsymbol{W}$ and $\boldsymbol{R}$ are two sets of parameters that we need to learn from two sets of examples $\boldsymbol{X}$ and $\boldsymbol{A}$ such that the inner products between their generated binary codes can preserve the inner products between the original data vectors.

Unlike previous works which predefine the form of hash function and learn from the data in a parametric way, *Discrete Graph Hashing* (DGH) learns the binary codes directly without involving the hash map. Specifically, they optimize the following objective function:

$$\begin{aligned} &\max_{\boldsymbol{B}} \ \operatorname{tr}(\boldsymbol{B}^T\boldsymbol{A}\boldsymbol{B}) \\ &s.t.\ \boldsymbol{B} \in \{1, -1\}^{n\times r}, \quad \boldsymbol{1}^T\boldsymbol{B} = \boldsymbol{0},\ \boldsymbol{B}^T\boldsymbol{B} = n\boldsymbol{I}_r, \end{aligned} \tag{5}$$

where $\boldsymbol{A}$ is a data-to-data affinity matrix obtained from anchor graph. However, as it further relies on anchor graph to approximate the pairwise similarity relationship and puts additional balance and orthogonal constraints on binary codes, the solution may further deviate from the the optimal one. In addition, as they use SSDM instead of DDDM, the pairwise distance may not be best preserved.

[1]It is common for preprocessing. Actually, to make DDDM feasible, we have to maintain a proper scale for comparing distances in the input space with distances in the Hamming space.

IV. Binary Multidimensional Scaling

A. Binary Codes Generation

Assume that each column of $\boldsymbol{X}$ has been normalized in the ℓ_2-norm. We propose a more elegant and natural way to solve the problem (3) and generate binary codes $\boldsymbol{Y}$ for training example $\boldsymbol{X}$. Unlike previous work which either explicitly constrains the form of hash map from $\boldsymbol{X}$ to $\boldsymbol{Y}$ or utilizes additional approximation like the anchor graph and the balance and orthogonality of binary code, we try to solve (3) directly. We wish to gain better solution by less relaxation or approximation to the original problem.

Since the objective function is quadratic with respect to binary variable $\boldsymbol{X}$, it is hard to solve directly. To overcome this difficulty, we propose to introduce the auxiliary variable $\mathbf{B}$. In this way, problem (3) could be rewritten as:

$$\begin{aligned} (\boldsymbol{Y}^*, \boldsymbol{B}^*) = \ &\operatorname{argmin}_{\boldsymbol{Y},\boldsymbol{B}} \ \|\boldsymbol{Y}^T\boldsymbol{B} - d\boldsymbol{X}^T\boldsymbol{X}\|_F^2, \\ &s.t.\ \boldsymbol{Y} = \boldsymbol{B}, \quad \boldsymbol{Y} \odot \boldsymbol{B} = \boldsymbol{E}. \end{aligned} \tag{6}$$

Here $\odot$ means the entry-wise matrix multiplication and $\boldsymbol{E}$ represents the matrix whose entries are all 1's. (6) is still hard to optimize due to its equality constraints. We may relax the problem as an unconstrained one with a penalty term over the violation of the equality constraints:

$$\begin{aligned} (\boldsymbol{Y}^*, \boldsymbol{B}^*) &\approx \operatorname{argmin}_{\boldsymbol{Y},\boldsymbol{B}} \mathcal{L}(\boldsymbol{Y}, \boldsymbol{B}; \lambda) \\ &= \frac{1}{2}\|\boldsymbol{Y}^T\boldsymbol{B} - d\boldsymbol{X}^T\boldsymbol{X}\|_F^2 \\ &\quad + \frac{\lambda}{2}\left(\|\boldsymbol{Y} - \boldsymbol{B}\|_F^2 + \|\boldsymbol{Y} \odot \boldsymbol{B} - \boldsymbol{E}\|_F^2\right). \end{aligned} \tag{7}$$

Note that as λ goes to $+\infty$, the optimal solution to (7) will coincide with the solution to (6). We propose to solve (7) directly by Alternating Minimization. It is an iterative procedure and will eventually converge to a binary solution as we gradually increase λ during the iterations. Specifically, each iteration consists of three steps:

1) Update $\mathbf{Y}$:

$$\mathbf{Y}_{i+1} = \operatorname{argmin}_{\mathbf{Y}} \mathcal{L}(\mathbf{Y}, \mathbf{B}_i; \lambda_i). \tag{8}$$

2) Update $\mathbf{B}$:

$$\mathbf{B}_{i+1} = \operatorname{argmin}_{\mathbf{B}} \mathcal{L}(\mathbf{Y}_{i+1}, \mathbf{B}; \lambda_i). \tag{9}$$

3) Update λ:

$$\lambda_{i+1} = \min(\rho\lambda_i, \lambda_{\max}). \tag{10}$$

The algorithm terminates when

$$\max\{\|\boldsymbol{Y}_{i+1} - \boldsymbol{Y}_i\|_\infty, \|\boldsymbol{B}_{i+1} - \boldsymbol{B}_i\|_\infty\} < \epsilon_1, \tag{11}$$

and

$$\max\{\|\boldsymbol{Y}_{i+1} - \boldsymbol{B}_{i+1}\|_\infty, \|\boldsymbol{Y}_{i+1} \odot \boldsymbol{B}_{i+1} - \boldsymbol{E}\|_\infty\} < \epsilon_2, \tag{12}$$

where $\|\cdot\|_\infty$ denotes the maximum absolute values of the entries in a matrix.

It is worth mentioning that all these steps have closed-form solutions. For example, in step 1, to achieve the minimum, the

gradient of the objective function with respect to $\boldsymbol{Y}$ must be zeros. That is,

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{Y}} = \boldsymbol{B}_i(\boldsymbol{B}_i^T\boldsymbol{Y} - d\boldsymbol{X}^T\boldsymbol{X}) + \lambda(\boldsymbol{Y} - 2\boldsymbol{B}_i + \boldsymbol{Y} \odot (\boldsymbol{B}_i \odot \boldsymbol{B}_i)) = 0. \quad (13)$$

Define $\boldsymbol{F}_i = \boldsymbol{B}_i\boldsymbol{B}_i^T + \lambda_i\boldsymbol{I}$, $\boldsymbol{G}_i = \lambda_i(\boldsymbol{B}_i \odot \boldsymbol{B}_i)$, and $\boldsymbol{H}_i = 2\lambda_i\boldsymbol{B}_i + d\boldsymbol{B}_i\boldsymbol{X}^T\boldsymbol{X}$. Denote the j-th column of $\boldsymbol{G}_i$, $\boldsymbol{H}_i$, $\boldsymbol{B}_i$, and $\boldsymbol{Y}$ as $\boldsymbol{g}_j$, $\boldsymbol{h}_j$, $\boldsymbol{b}_j$, and $\boldsymbol{y}_j$, respectively. It is easy to see that $\boldsymbol{y}_j$ is the solution to the equation

$$(\boldsymbol{F}_i + \mathrm{diag}(\boldsymbol{g}_j))\boldsymbol{y} = \boldsymbol{h}_j. \quad (14)$$

Namely, $\boldsymbol{y}_j = (\boldsymbol{F}_i + \mathrm{diag}(\boldsymbol{g}_j))^{-1}\boldsymbol{h}_j$. In a similar way, one can get n columns of $\boldsymbol{Y}_{i+1}$ by solving n linear equations. Since these equations are independent, it is easy to accelerate the algorithm by parallelization when n is large. Analogously, we can obtain the solution in step 2. We may gradually increase λ to impose the binary constraint in step 3. The above procedure will eventually converge to a local minimum [1] with $\boldsymbol{Y}$ and $\boldsymbol{B}$ randomly initialized, as the objective function is smooth.

During the experiments, we set $\epsilon_1 = \epsilon_2 = 0.01$, $\lambda_0 = 0.5$, $\rho = 1.5$, and $\lambda_{max} = 10^5$.

B. Out-of-Sample Extension

Given the database $\boldsymbol{X}$ and its binary hash codes $\boldsymbol{Y}$, we get the optimal binary code $\boldsymbol{y}$ for an out-of-sample query $\boldsymbol{x}$ by minimizing the similar objective function:

$$\boldsymbol{y}^* = \mathrm{argmin}_{\boldsymbol{y}} \sum_i \left(\frac{1}{2}\|\boldsymbol{x} - \boldsymbol{x}_i\|_2^2 - \frac{1}{2d}\|\boldsymbol{y} - \boldsymbol{y}_i\|_2^2\right)^2 = \mathrm{argmin}_{\boldsymbol{y}} \|\boldsymbol{Y}^T\boldsymbol{y} - d\boldsymbol{X}^T\boldsymbol{x}\|_2^2, \quad s.t.\ \boldsymbol{y} \in \{-1, 1\}^d. \quad (15)$$

Let $\boldsymbol{Y}^T = \boldsymbol{Q}\boldsymbol{S}$ be the polar decomposition of $\boldsymbol{Y}^T$, where $\boldsymbol{Q}$ is column orthonormal and $\boldsymbol{S} = (\boldsymbol{Y}\boldsymbol{Y}^T)^{1/2}$ is positive-semidefinite. Denote $\boldsymbol{b} = \boldsymbol{X}^T\boldsymbol{x}$, and $\tilde{\boldsymbol{b}} = d\boldsymbol{Q}^T\boldsymbol{b}$, (15) is equivalent to:

$$\boldsymbol{y}^* = \mathrm{argmin}_{\boldsymbol{y}} \|\boldsymbol{S}\boldsymbol{y} - \tilde{\boldsymbol{b}}\|_2^2, \quad s.t.\ \boldsymbol{y} \in \{-1, 1\}^d. \quad (16)$$

Solving (16) is also NP-hard in general. However, if $\boldsymbol{S}$ is a diagonal matrix whose entries are all positive, the solution is simply given as $\boldsymbol{y}^* = \mathrm{sgn}(\tilde{\boldsymbol{b}})$. Actually, under mild assumptions below, we can show in Lemma 1 that $\boldsymbol{S}$ is close to a diagonal matrix with extremely high probability when n, the number of training examples, is large.

Assumption 1: Assume that each entry of $\boldsymbol{Y}$ is an i.i.d. Bernoulli random variable, so that $\mathcal{P}(Y_{ij} = 1) = \mathcal{P}(Y_{ij} = -1) = \frac{1}{2}$ for any $i \in \{1, 2, \cdots, d\}$, and $j \in \{1, 2, \cdots, n\}$.

The independence among the bits of hash codes is natural if we want all available hash codes to be uniformly distributed, and it may not harm the preservation of the distances among features. Note that it does not mean that *given a feature* the bits of its hash code can be arbitrary.

Lemma 1: Based on Assumption 1, $|S_{ii} - \sqrt{n}| \le C_{1\epsilon}\sqrt{n}$ and $|S_{ij}| \le C_{2\epsilon}\sqrt{n}$ $(i \ne j)$ hold with probability at least $1 - d(d-1)\exp\left(-\frac{n\epsilon^2}{2}\right)$, where $C_{1\epsilon} = c(d-1)^2\epsilon^2$, $C_{2\epsilon} = c(d-1)^2\epsilon^2 + \frac{1}{2}\epsilon$, and c is a constant that makes $|\sqrt{1+\gamma} - 1 - \frac{1}{2}\gamma| \le c\gamma^2$ true when $|\gamma| \le (d-1)\epsilon$.

The proofs in this section can be found in appendix. Next, we point out that even if $\boldsymbol{S}$ is not strictly diagonal, we have $y_i^* = \mathrm{sgn}(\tilde{b}_i)$ if $|\tilde{b}_i|$ is "large enough".

Lemma 2: Let $\boldsymbol{y}^*$ be the optimal solution to (16)*. If*

$$|\tilde{b}_i| > C_{0i} \triangleq \frac{\sum_{j\ne i}(\sum_{k\ne i}|S_{jk}| + |\tilde{b}_j|)|S_{ij}|}{S_{ii}} + \sum_{k\ne i}|S_{ik}|, \quad (17)$$

then the i-th bit of $\boldsymbol{y}^*$ must be $\mathrm{sgn}(\tilde{b}_i)$.

Note that distinguishing whether $|\tilde{b}_i|$ is "large enough" is meaningful because when $|\tilde{b}_i|$ is small, its sign can be considered as ambiguous. Both the Lemmas above suggest that $\mathrm{sgn}(\tilde{\boldsymbol{b}})$ might be a good approximate solution to (16). However, the quantized least squares (QLS) solution

$$\hat{\boldsymbol{y}}^* = \mathrm{sgn}\left(\boldsymbol{S}^{-1}\tilde{\boldsymbol{b}}\right) = \mathrm{sgn}\left(\boldsymbol{Y}^+\boldsymbol{X}^T\boldsymbol{x}\right), \quad (18)$$

where $\boldsymbol{Y}^+$ is the pseudo-inverse of $\boldsymbol{Y}$, may be more natural as the solution to (16) is $\boldsymbol{S}^{-1}\tilde{\boldsymbol{b}}$ if the binary constraint is removed. The rationality and error bound of the QLS solution are provided in Lemma 3 and Theorem 1, respectively. Namely, $\mathrm{sgn}\left(\tilde{b}_i\right) = \mathrm{sgn}\left((\boldsymbol{S^{-1}}\tilde{\boldsymbol{b}})_i\right)$ holds with large probability as the number of example points increases, and the difference between the objective function values with $\mathrm{sgn}(\tilde{\boldsymbol{b}})$ and global optimal could be upper bounded, so that the error between the QLS solution and the global optimal solution could be estimated.

Lemma 3: Based on Assumption 1, if $\tilde{b}_i$ satisfies (17) *and $\epsilon < \frac{1}{2(d-1)}$, then $\mathrm{sgn}\left(\tilde{b}_i\right) = \mathrm{sgn}\left((\boldsymbol{S^{-1}}\tilde{\boldsymbol{b}})_i\right)$ holds with probability at least $1 - 2d(d-1)\exp(-\frac{n\epsilon^2}{2}) - 2d\exp\left(-\frac{C_{0i}^2}{2\left[(d-1)(2d-1)\tilde{C}_{3\epsilon}\epsilon\right]^2}\right)$, where $\tilde{C}_{3\epsilon} = d(1 + 2d(d-1)^2\epsilon^2 + \frac{(d-1)\epsilon}{2})$.*

Theorem 1: Based on Assumption 1, if the first r bits in $\tilde{\boldsymbol{b}}$ satisfy (17) *and $\epsilon < \frac{1}{2(d-1)}$, then with probability at least $1 - 2d(d-1)r\exp(-\frac{n\epsilon^2}{2}) - 2d\sum_{i=1}^r \exp\left(-\frac{C_{0i}^2}{2\left[(d-1)(2d-1)\tilde{C}_{3\epsilon}\epsilon\right]^2}\right)$, the difference between the objective function values with the QLS solution and the global optimal solution $\boldsymbol{y}^*$ is upper bounded by*

$$4\sum_i\left(\sum_k|S_{ik}| + \left|\tilde{b}_i\right|\right)\left(\sum_{k>r}|S_{ik}|\right) - 4\sum_{i>r}S_{ii}^2, \quad (19)$$

where $\tilde{C}_{3\epsilon}$ is the same as in Lemma 3.

Note that in real problems, C_{0i} is usually of the magnitude of $O(\epsilon\sqrt{n})$, thus the probability in Lemma 3 and Theorem 1 is very close to 1 when n is large. Moreover, the computational complexity of QLS solution is no more expensive than taking the sign of $\tilde{\boldsymbol{b}}$ directly as explicitly calculating $\tilde{\boldsymbol{b}}$ relies on the polar decomposition of $\boldsymbol{Y}^T$. Besides, the adoption of the QLS solution allows an online learning extension, which will be shown later. One should also be reminded that in practice $\boldsymbol{A} \triangleq \boldsymbol{Y}^+\boldsymbol{X}^T$ can be pre-computed. Thus for out-of-sample query $\boldsymbol{x}_{new}$, its hash code can be easily obtained at a low cost, by $\boldsymbol{y}_{new} = \mathrm{sgn}(\boldsymbol{A}\boldsymbol{x}_{new})$.

One should be reminded that we only use linear perceptrons as hash map in out-of-sample query stage as a result of theoretical analysis. The hash codes for the whole database and training set are generated by alternating minimization, without confining the form of hash function. This distinguishes our work from previous ones.

C. BMDS With Sampling

Our method is scalable for big data at the scale of millions. For truly large datasets, we may resort to the sampling technique to accelerate the training stage and save the memory. Specifically, we randomly select a small portion of the examples (10% in our experiments) in the database as the training set. Then the remaining examples are treated as out-of-sample queries, whose codes are predicted by (18). We will show that significant acceleration can be achieved by doing so without sacrificing much accuracy.

D. Online Learning Extension

Our method also enables a natural online learning extension inspired by QLS. Suppose that we have already obtained n examples in the databset and we have learned binary codes $\boldsymbol{Y}_n$ from the data matrix $\boldsymbol{X}_n$. Then we map the new query $\boldsymbol{x}_{n+1}$ to binary code $\boldsymbol{y}_{n+1}$ by

$$\boldsymbol{y}_{n+1} = \operatorname{sgn}((\boldsymbol{Y}_n\boldsymbol{Y}_n^T)^{-1}\boldsymbol{Y}_n\boldsymbol{X}_n^T\boldsymbol{x}_{n+1}) \triangleq \operatorname{sgn}(\boldsymbol{A}_n\boldsymbol{x}_{n+1}). \quad (20)$$

Similarly, we have

$$\boldsymbol{y}_{n+2} = \operatorname{sgn}(\boldsymbol{A}_{n+1}\boldsymbol{x}_{n+2}), \quad (21)$$

where $\boldsymbol{A}_{n+1} = (\boldsymbol{Y}_{n+1}\boldsymbol{Y}_{n+1}^T)^{-1}\boldsymbol{Y}_{n+1}\boldsymbol{X}_{n+1}^T$. In fact, there exists an efficient way to calculate $\boldsymbol{A}_{n+1}$ recursively. Define $\boldsymbol{Z}_n = (\boldsymbol{Y}_n\boldsymbol{Y}_n^T)^{-1}$. According to the Sherman-Morrison formula [25], noting that $\boldsymbol{Y}_{n+1} = [\boldsymbol{Y}_n \quad \boldsymbol{y}_{n+1}]$ and $\boldsymbol{X}_{n+1} = [\boldsymbol{X}_n \quad \boldsymbol{x}_{n+1}]$, we have

$$\begin{aligned}\boldsymbol{Z}_{n+1} &= (\boldsymbol{Y}_{n+1}\boldsymbol{Y}_{n+1}^T)^{-1} \\ &= (\boldsymbol{Y}_n\boldsymbol{Y}_n^T + \boldsymbol{y}_n\boldsymbol{y}_n^T)^{-1} \\ &= \boldsymbol{Z}_n - \frac{(\boldsymbol{Z}_n\boldsymbol{y}_{n+1})(\boldsymbol{y}_{n+1}^T\boldsymbol{Z}_n)}{1+\boldsymbol{y}_{n+1}^T\boldsymbol{Z}_n\boldsymbol{y}_{n+1}}, \quad (22)\\ \boldsymbol{A}_{n+1} &= \boldsymbol{Z}_{n+1}\boldsymbol{Y}_{n+1}\boldsymbol{X}_{n+1}^T \\ &= (\boldsymbol{Z}_n - \frac{\boldsymbol{Z}_n\boldsymbol{y}_{n+1}\boldsymbol{y}_{n+1}^T\boldsymbol{Z}_n}{1+\boldsymbol{y}_{n+1}^T\boldsymbol{Z}_n\boldsymbol{y}_{n+1}})(\boldsymbol{Y}_n\boldsymbol{X}_n^T + \boldsymbol{y}_{n+1}\boldsymbol{x}_{n+1}^T) \\ &= \boldsymbol{Q}_{n+1} - \frac{(\boldsymbol{Z}_n\boldsymbol{y}_{n+1})(\boldsymbol{y}_{n+1}^T\boldsymbol{Q}_{n+1})}{1+\boldsymbol{y}_{n+1}^T\boldsymbol{Z}_n\boldsymbol{y}_{n+1}}, \quad (23)\end{aligned}$$

where $\boldsymbol{Q}_{n+1} = \boldsymbol{A}_n + (\boldsymbol{Z}_n\boldsymbol{y}_{n+1})\boldsymbol{x}_{n+1}^T$. Given $\boldsymbol{A}_n$ and $\boldsymbol{Z}_n$, it is easy to see that we can get $\boldsymbol{A}_{n+1}$ and $\boldsymbol{Z}_{n+1}$ by only several matrix-vector multiplications, which are efficient.

In practice, we can generate the binary codes for the first data chunk by Alternating Minimization, as introduced before. Then we obtain the codes for the rest of the data chunks in an online learning fashion as we have discussed in this section. We will adopt this mechanism for experiments.

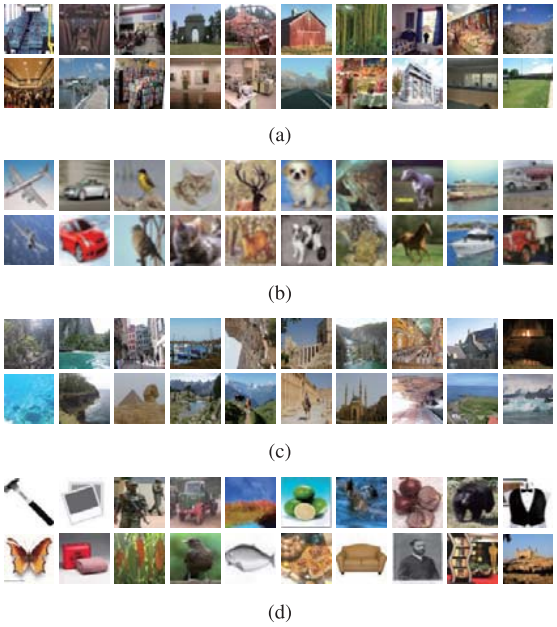


Fig. 1. Example images from SUN-397, CIFAR-10, GIST-1M, and TINY-1M, respectively. (a) SUN-397. (b) CIFAR-10. (c) GIST-1M. (d) TINY-1M.

E. Complexity Analysis

1) Training: We employ an iterative method for training. In each iteration, we alternately update the matrix $\boldsymbol{Y}$ and $\boldsymbol{B}$ in a similar way. That is to say, we need to do a big matrix multiplication plus solving n independent linear equations of size d. Thus the total computational complexity is $O(nd^3 + nmd)$ in each iteration. Note that n linear equations are independent of each other and can be solved in parallel. Advanced linear equation solvers could accelerate the algorithm further. In addition, the sampling technique that is introduced before could significantly reduce the training time. Thus our algorithm is scalable to large scale datasets.

2) Out-of-Sample Coding: As we can see, the time complexity of out-of-sample coding is $O(ndm + nd^2 + d^3) + O(md)$. The former O results from pre-computing matrix $\boldsymbol{A}$ *once*, and the latter is the time cost for each out-of-sample query, which is the same as those hashing methods which rely on linear perceptron mapping. One could further resort to the sampling technique to accelerate the computation of $\boldsymbol{A}$.

3) Online Learning: As mentioned above, our online learning scheme only involves several matrix-vector multiplications for updating $\boldsymbol{A}_{n+1}$, and *one* matrix-vector multiplication for the generating hash code for the new query. So the total computational complexity is $O(md)$, which is very cheap.

V. Experiments

We conduct extensive experiments to verify the superiority of our method in terms of accuracy and efficiency, in both batch and online fashions. We divide the datasets such that 95% examples are regarded as the training database, and the remaining are the out-of-sample queries. Examples are zero-centered and ℓ_2 normalized. For each query, we use its top 1% nearest neighbors based on the ℓ_2 distances in

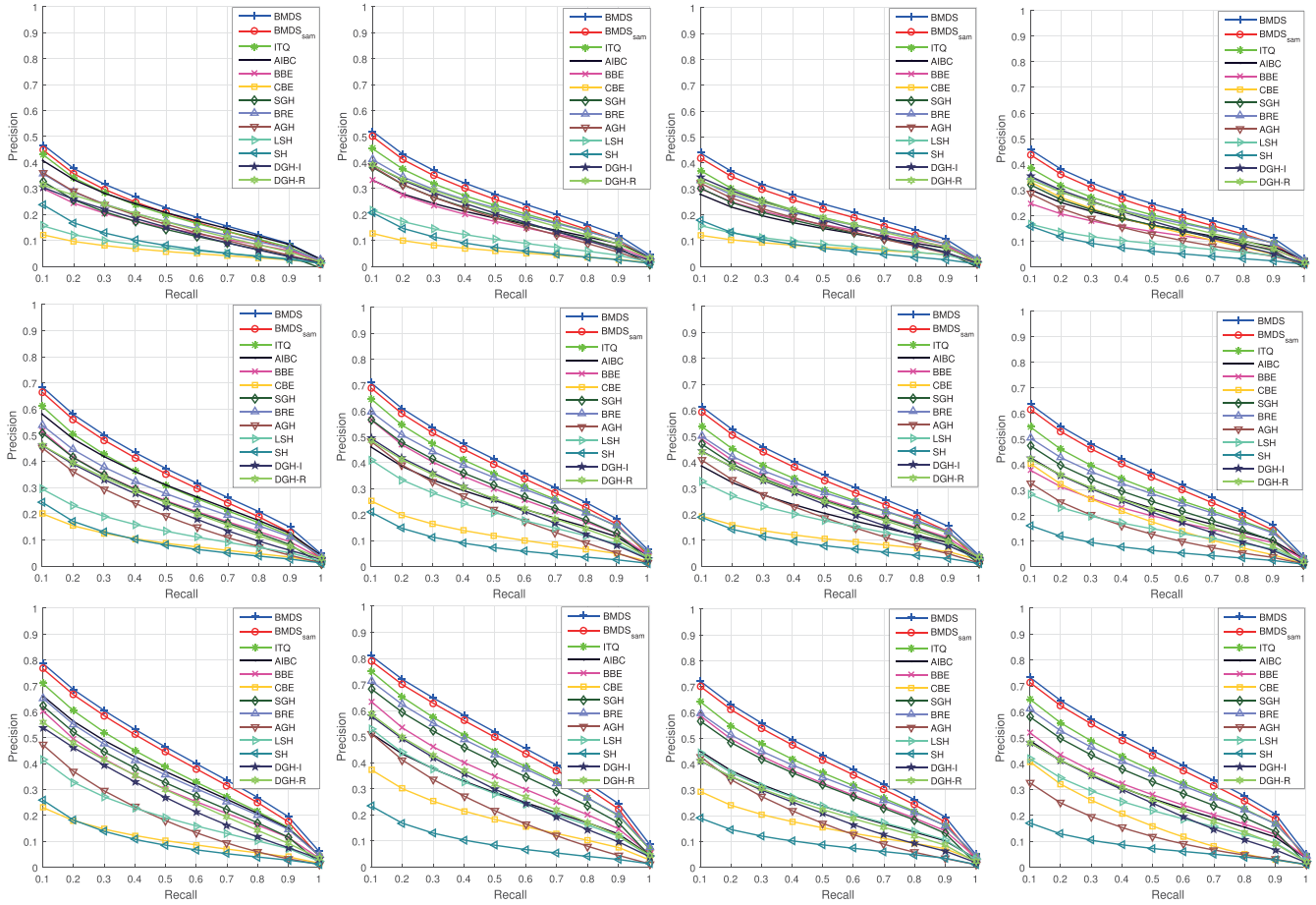


Fig. 2. Precision-Recall curves (from top to bottom) with 32-bit, 64-bit, and 96-bit hash codes on **CIFAR-10**, **SUN-397**, **TINY-1M**, and **GIST-1M** (from left to right), respectively. "BMDS$_{\text{sam}}$" is BMDS with sampling. Best viewed in color.

the original feature space as the ground truth. We evaluate our method based on the Hamming ranking scheme, and use precision, recall, and mean average precision (mAP) as the evaluation metric. We repeat the experiments 10 times with different datasets splitting, initialization and sampling. We report average scores in this section. All experiments are done on a server with an Intel Corporation Xeon E5 2.6GHz CPU and a 128GB RAM.

We evaluate our proposed Binary Multidimensional Scaling (BMDS) method on several benchmark datasets for approximate nearest neighbor search:

- **SUN-397** [37] contains 60K images with a large variety of scenes. A 512 dimensional GIST [6] feature is employed to represent each image.
- **CIFAR-10** [16] consists of 60K tiny images in 10 classes, each represented by a 512 dimensional GIST feature vector.
- **GIST-1M** [14] is a standard evaluation set to judge the quality of approximate nearest neighbor search. The database contains 1M images, mostly combined from Holidays dataset and Flickr1M dataset. Each image is represented by a 960-dim GIST descriptor.
- **TINY-1M** [32] is a subset of Tiny Images Dataset, which presents a visualization of all the nouns in English. TINY-1M contains one million images. Each image is represented by a 384-dim GIST descriptor.

Example images from each dataset are shown in Figure 1.

A. Comparison With Batch Based Methods

In this section, we compare our results with representative batch based unsupervised hashing methods, i.e. LSH, SH, ITQ, AIBC, BBE, CBE, BRE, DGH,[2] SGH, and AGH, respectively. For DGH and AIBC, we implement the algorithm according to their paper. For other methods, we use the source code provided by their authors. We tune the parameters as suggested by their authors.

Figure 2 shows the precision-recall curves of different algorithms on various datasets, with 32, 64, and 96 bits, respectively. Figure 3 shows the precision and recall[3] curves of different algorithms on different datasets, with 96-bit hash codes. These results suggest that BMDS consistently outperforms the state-of-the-art methods in terms of the accuracy in approximate nearest neighbor search. It also suggests that we will not suffer from much loss by random sampling. By directly generating the hash codes without predefining the form of hash map, we reduce the loss brought by the

[2]With different initialization, there are two versions of DGH, namely DGH-I and DGH-R. See [21] for details.

[3]The number of retrieved examples is set to be 1 to 10 times of the ground-truth nearest neighbor examples when evaluating recall rates so that the score could be maintained in a proper scale.

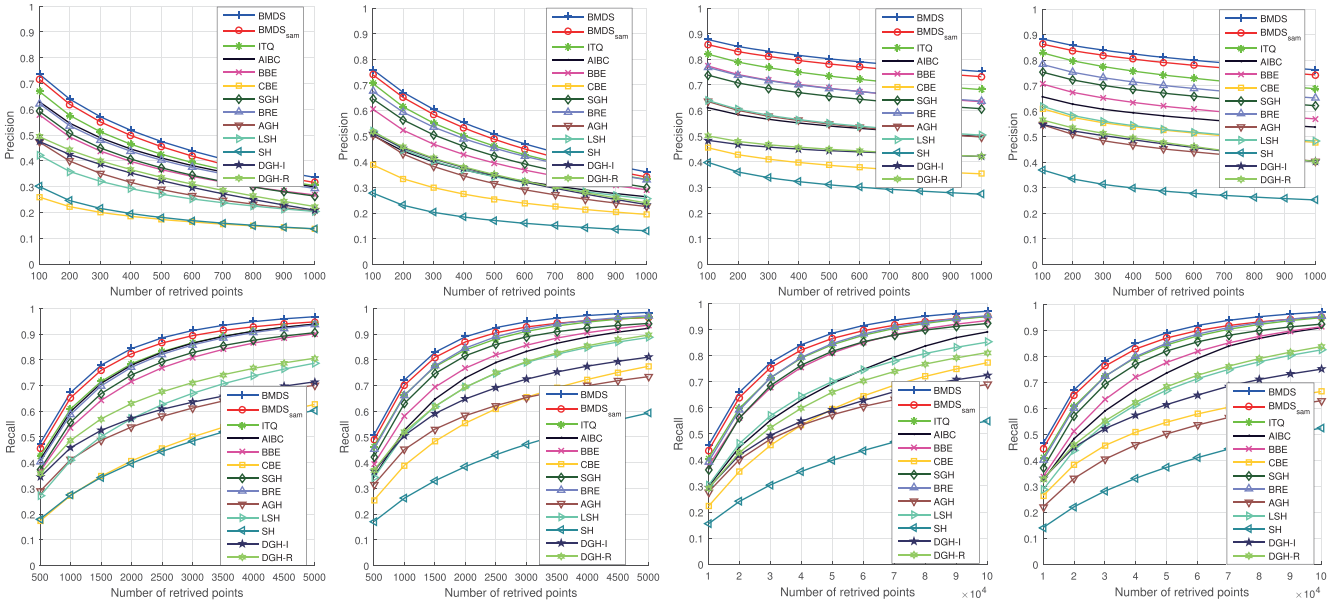


Fig. 3. Precision curves and recall curves (from top to bottom) with 96-bit hash codes on **CIFAR-10**, **SUN-397**, **TINY-1M**, and **GIST-1M** (from left to right), respectively. "BMDS$_{\text{sam}}$" is BMDS with sampling. Best viewed in color.

TABLE I

HAMMING RANKING PERFORMANCE ON **GIST-1M** AND **TINY-1M**. d DENOTES THE NUMBER OF HASH BITS USED IN THE HASHING METHODS. "BMDS SAMPL." IS BMDS WITH SAMPLING. ALL TRAINING AND AVERAGE TEST TIMES ARE IN SECONDS

Type	Method	GIST-1M					Tiny-1M				
		mAP (std)			Train time	AveTest time	mAP (std)			Train time	AveTest time
		$d=32$	$d=64$	$d=96$	$d=64$	$d=64$	$d=32$	$d=64$	$d=96$	$d=64$	$d=64$
Ours	BMDS	**0.2784** (0.0020)	**0.3973** (0.0018)	**0.4675** (0.0013)	968.72	1.8 E −6	**0.2501** (0.0019)	**0.3647** (0.0015)	**0.4433** (0.0014)	837.21	1.1 E −6
	BMDS sampl.	0.2273 (0.0030)	0.3401 (0.0026)	0.4225 (0.0025)	120.97	1.8 E −6	0.2114 (0.0032)	0.3384 (0.0025)	0.4107 (0.0025)	107.24	1.1 E −6
Linear func.	ITQ [12]	0.2244 (0.0044)	0.3277 (0.0041)	0.3980 (0.0040)	54.80	1.8 E −6	0.2125 (0.0044)	0.3166 (0.0042)	0.3852 (0.0041)	49.01	1.2 E −6
	AIBC [26]	0.1856 (0.0006)	0.2544 (0.0010)	0.2950 (0.0018)	929.76	1.8 E −6	0.1667 (0.0008)	0.2259 (0.0011)	0.2575 (0.0016)	828.75	1.1 E −6
	BBE [10]	0.1566 (0.0044)	0.2294 (0.0043)	0.3275 (0.0043)	453.32	3.0 E −6	0.1765 (0.0045)	0.3103 (0.0043)	0.3455 (0.0044)	380.17	2.3 E −6
	CBE [38]	0.0614 (0.0039)	0.1265 (0.0041)	0.1681 (0.0040)	804.37	3.6 E −6	0.0795 (0.0041)	0.1283 (0.0040)	0.1751 (0.0042)	202.76	2.8 E −6
Ker. func.	SGH[15]	0.1866 (0.0018)	0.2864 (0.0019)	0.3360 (0.0019)	288.73	9.2 E −6	0.1972 (0.0020)	0.2681 (0.0021)	0.3352 (0.0020)	258.12	8.1 E −6
	BRE [17]	0.2145 (0.0021)	0.3129 (0.0019)	0.3752 (0.0020)	1388.54	8.9 E −6	0.2042 (0.0023)	0.3121 (0.0022)	0.3751 (0.0020)	1269.58	7.9 E −6
	AGH [23]	0.1834 (0.0015)	0.2092 (0.0015)	0.2188 (0.0016)	180.03	9.1 E −6	0.1757 (0.0016)	0.2101 (0.0017)	0.2244 (0.0014)	163.21	7.9 E −6
	DGH-I [21]	0.2086 (0.0013)	0.2625 (0.0014)	0.3012 (0.0012)	840.23	9.2 E −6	0.1906 (0.0018)	0.2365 (0.0016)	0.2740 (0.0017)	766.46	8.0 E −6
	DGH-R [21]	0.2208 (0.0015)	0.2670 (0.0016)	0.3175 (0.0013)	905.36	9.2 E −6	0.1959 (0.0016)	0.2463 (0.0014)	0.2884 (0.0016)	802.38	8.0 E −6
Rand. proj.	LSH [9]	0.0963 (0.0085)	0.1584 (0.0088)	0.2413 (0.0092)	1.46	1.8 E −6	0.0988 (0.0083)	0.2047 (0.0089)	0.2586 (0.0093)	1.28	1.1 E −6
Eigen. func.	SH [36]	0.0815 (0.0000)	0.0868 (0.0000)	0.0919 (0.0000)	21.55	1.1 E −5	0.0946 (0.0000)	0.1035 (0.0000)	0.1102 (0.0000)	17.90	9.4 E −6

error-prone hypothesis on hash functions, thus we preserve the pairwise distances better.

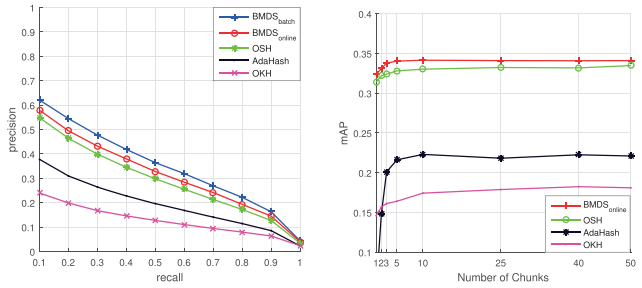
As suggested above, our algorithm is also scalable to large scale datasets. To further verify this claim, we report the experimental results on two large datasets, namely **GIST-1M** and **TINY-1M**, in Table I. One should be reminded that in real-world applications, people care more about the average test time for out-of-sample query, and our method is among the fastest in this aspect. As for the training time, although some methods are more efficient in training, they suffer from over-simplified models that prevent them from achieving high accuracy results. For example, LSH is based on data-independent random projection, so the pairwise distances cannot be preserved well. SH relies on learning eigenfunctions,

Fig. 4. Example image retrieval results on CIFAR-10 (Top@20). Each method uses 96 bits codes. The false positive results are remarked with red rectangles. The query images are shown at the first row. Best viewed in color.

whose valid information concentrates on particular hash bits. ITQ, CBE, BBE, and AIBC use linear perceptrons as the hash map, which may lead to high quantization error. The efficiency of AGH, SGH, DGH, and BRE rely on the choice of anchor points. On the other hand, although our methods are not the fastest in the training stage, they are not the slowest either. Our algorithm could be further accelerated with proper parallel implementation. Besides, the sampling technique[4] can significantly reduce the training time without losing much accuracy. As shown in Table I, the training time of BMDS with sampling is competitive to most methods, while its mAP still exceeds others.

Finally, we provide two example queries on CIFAR-10, namely *airplane* and *frog*, to qualitatively evaluate the performance. Each hash method uses 96-bit codes, and the top 20 returned results based on Hamming ranking are displayed in Figure 4. For *airplane*, examples from other class like bird or ship may interfere with the results due to the similar appearance. For *frog*, the noisy background might be a challenge. It could be observed that most compared methods fail in either of two ways: 1) They falsely map the examples that are far away in original space to similar hash codes. 2) They fail to further recognize the top nearest-neighbors.

[4]Note that those kernel-based methods have already employed the sampling technique, namely choosing the anchor points.



(a) (b)

Fig. 5. Retrieval performance of online methods on **GIST-1M** with 64-bit hash codes.

TABLE II
AVERAGE TRAINING TIME (SECONDS/CHUNK) OF DIFFERENT ONLINE LEARNING METHODS ON **GIST-1M**

Method	Time
OKH [13]	11.25
OSH [18]	3.22
AdaHash [3]	43.16
Online BMDS	12.66

For example, they may not be able to further pick up the top 1‰ nearest-neighbors from top hundredth. By utilizing the proposed scheme, our algorithm could best preserve the pairwise distance and reduce the failure in both scenarios. We could easily see that in both queries, our algorithm achieves the highest accuracy with almost no false-positives.

B. Comparison With Online Hashing Methods

In this experiment, we focus on learning hash codes online for **GIST-1M**. We divide the training set into 50 chunks with nearly 20K examples per chunk. The data become available as a stream to simulate the real-world scenario. We compare the proposed method with the state-of-the-art online hashing approaches, OKH, OSH, and AdaHash. We use the source code from their authors and tune the parameters according to their guidance. We use mAP, based on the Hamming ranking, as the evaluation metric. Experimental results are shown in Figure 5 and Table II, respectively.

As we can see, the precision-recall curve in Figure 5(a) suggests that our approach outperforms the state-of-the-art methods in terms of accuracy. The results of online BMDS are even competitive to those of batch BMDS. Figure 5(b) shows that our algorithm rapidly converges to a good solution after a few iterations. It turns out that we will benefit a lot if we associate the learning of current data chunk with chunks that we have already learned. In addition, since updating hash map only involves several matrix-vector multiplications when a new example comes, it is efficient as illustrated in Table II, hence it is a good alternative for processing massive data.

VI. CONCLUSION

We propose an integrated framework for unsupervised hash codes learning that could work in both batch and online settings. In the batch mode, our algorithm tries to learn the binary codes directly based on the ℓ_2 distances in the original feature space without predefining the form of hash function. In the online mode, we take the holistic distance relationship between current query example and the database into account, rather than considering only the current data chunk. Both theoretical and experimental results demonstrate the advantage of our approach in distance preservation and potential usage for real-world big data problems.

APPENDIX A
PROOF OF LEMMA 1

First of all, we discuss the property of Gram matrix $\boldsymbol{G}^Y$.

Proposition 1: Based on Assumption 1, $\left|G^Y_{ij}\right| < n\epsilon$ *holds for every* $i \neq j$ *with probability at least* $1 - d(d-1)\exp\left(-\frac{n\epsilon^2}{2}\right)$.

Proof: Since $\left|G^Y_{ij}\right| = \left|Y_{\cdot i}Y^T_{\cdot j}\right|$, namely the inner product of the i-th row and the j-th row of $\boldsymbol{Y}$. With the assumption that all the entries of $\boldsymbol{Y}$ are independent Bernoulli variables, we could deduce from the Chernoff-Hoeffding inequality that

$$\mathcal{P}\left(\left|G^Y_{ij}\right| > n\epsilon\right) \leq 2\exp\left(-\frac{n\epsilon^2}{2}\right). \tag{24}$$

Since there are in total $\frac{d(d-1)}{2}$ different off-diagonal entries in $\boldsymbol{G}^Y$, we can conclude that with probability at least $1 - d(d-1)\exp\left(-\frac{n\epsilon^2}{2}\right)$, $\left|G^Y_{ij}\right| < n\epsilon$ $(\forall i \neq j)$. □

We now give the proof to Lemma 1. *Proof:* Suppose $\left|G^Y_{ij}\right| < n\epsilon$ holds for every $i \neq j$. Consider the eigenvalue decomposition of $\boldsymbol{G}^Y$: $\boldsymbol{G}^Y = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^{\boldsymbol{T}}$, where $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \cdots, \lambda_n)$. Then from $G^Y_{ii} = n$ and $\left|G^Y_{ij}\right| \leq n\epsilon$ $(\forall i \neq j)$, we have

$$\sum_k \lambda_k U^2_{ik} = n, \tag{25}$$

$$\left|\sum_k \lambda_k U_{ik}U_{jk}\right| \leq n\epsilon, \quad (\forall i \neq j). \tag{26}$$

Furthermore, by the orthogonality of $\boldsymbol{U}$, we have

$$\sum_k U^2_{ik} = 1, \tag{27}$$

$$\sum_k U_{ik}U_{jk} = 0, \quad (\forall i \neq j). \tag{28}$$

Finally, by the Gershgorin discs theorem all eigenvalues of G^Y lie in an interval centered at n:

$$|\delta_i| \leq (d-1)n\epsilon, \tag{29}$$

where $\delta_i = \lambda_i - n$. By (25) and (27), we have

$$\sum_k \delta_k U^2_{ik} = \sum_k (\lambda_k - n)U^2_{ik} = \sum_k \lambda_k U^2_{ik} - n\sum_k U^2_{ik} = 0. \tag{30}$$

By (26) and (28), we have

$$\left|\sum_k \delta_k U_{ik}U_{jk}\right| = \left|\sum_k (\lambda_k - n)U_{ik}U_{jk}\right| \leq n\epsilon, \quad \forall i \neq j. \tag{31}$$

From (27), (30), and the fact that $\left|\sqrt{1+\gamma}-1-\frac{1}{2}\gamma\right| \le c\gamma^2$, we have

$$\begin{aligned}
\left|S_{ii}-\sqrt{n}\right| &= \left|\sum_k \sqrt{\lambda_k}U_{ik}^2-\sqrt{n}\right| \\
&= \left|\sum_k \left(\sqrt{n+\delta_k}-\sqrt{n}-\frac{1}{2}n^{-\frac{1}{2}}\delta_k\right)U_{ik}^2\right| \\
&\le \sum_k \left|\sqrt{n+\delta_k}-\sqrt{n}-\frac{1}{2}n^{-\frac{1}{2}}\delta_k\right| U_{ik}^2 \\
&\le \sum_k cn^{-\frac{3}{2}}\delta_k^2 U_{ik}^2 \\
&\le c\sqrt{n}(d-1)^2\epsilon^2. \qquad (32)
\end{aligned}$$

Similarly, $\forall i \ne j$,

$$\begin{aligned}
\left|S_{ij}\right| &= \left|\sum_k \sqrt{n+\delta_k}U_{ik}U_{jk}\right| \\
&= \left|\sum_k (\sqrt{n+\delta_k}-\sqrt{n}-\frac{1}{2}n^{-\frac{1}{2}}\delta_k)U_{ik}U_{jk}\right. \\
&\quad \left.+\frac{1}{2}n^{-\frac{1}{2}}\sum_k \delta_k U_{ik}U_{jk}\right| \\
&\le \sum_k \left|\sqrt{n+\delta_k}-\sqrt{n}-\frac{1}{2}\sqrt{n}\delta_k\right| |U_{ik}||U_{jk}| \\
&\quad +\frac{1}{2}\sqrt{n}\left|\sum_k \delta_k U_{ik}U_{jk}\right| \\
&\le \sum_k cn^{-\frac{3}{2}}\delta_k^2|U_{ik}||U_{jk}|+\frac{1}{2}\sqrt{n}\epsilon \\
&\le c\sqrt{n}(d-1)^2\epsilon^2+\frac{1}{2}\sqrt{n}\epsilon. \qquad (33)
\end{aligned}$$

Since $\left|G_{ij}^Y\right| < n\epsilon$ holds for every $i \ne j$ with probability at least $1-d(d-1)\exp\left(-\frac{n\epsilon^2}{2}\right)$, (32) and (33) hold with at least the same probability. □

It is easy to see that we have similar properties of $\mathbf{\Phi} = \boldsymbol{S}^{-1}$.

Proposition 2: Based on Assumption 1, $\left|\Phi_{ii}-\frac{1}{\sqrt{n}}\right| \le \frac{\tilde{C}_{1\epsilon}}{\sqrt{n}}$ *and* $\left|\Phi_{ij}\right| \le \frac{\tilde{C}_{2\epsilon}}{\sqrt{n}}$ *hold with probability at least* $1-d(d-1)\exp\left(-\frac{n\epsilon^2}{2}\right)$, *where* $\tilde{C}_{1\epsilon}=\tilde{c}(d-1)^2\epsilon^2$, $\tilde{C}_{2\epsilon}=\tilde{c}(d-1)^2\epsilon^2+\frac{1}{2}\epsilon$, *and* $\tilde{c}$ *is a constant that makes* $\left|\frac{1}{\sqrt{1+\gamma}}-1+\frac{1}{2}\gamma\right| \le \tilde{c}\gamma^2$ *true when* $|\gamma| \le (d-1)\epsilon$.

Appendix B
Proof of Lemma 2

Proof: If the i-th bit of $\boldsymbol{y}^*$ is $-\operatorname{sgn}(\tilde{b}_i)$, we define a new vector $\tilde{\boldsymbol{y}}$ whose bits are identical to those of $\boldsymbol{y}^*$ except the i-th. We will show that the objective function value with $\tilde{\boldsymbol{y}}$ is lower than that of $\boldsymbol{y}^*$, making a contradiction on the optimality of $\boldsymbol{y}^*$. Actually, the difference between the two objective function values is:

$$\begin{aligned}
&\sum_{j\ne i}\left(\sum_{k\ne j,i} S_{jk}y_k^* + S_{jj}y_j^* + S_{ji}\operatorname{sgn}(\tilde{b}_i)-\tilde{b}_j\right)^2 \\
&\quad +\left(\sum_{k\ne i} S_{ik}y_k^* + S_{ii}\operatorname{sgn}(\tilde{b}_i)-\tilde{b}_i\right)^2 \\
&\quad -\sum_{j\ne i}\left(\sum_{k\ne j,i} S_{jk}y_k^* + S_{jj}y_j^* - S_{ji}\operatorname{sgn}(\tilde{b}_i)-\tilde{b}_j\right)^2 \\
&\quad -\left(\sum_{k\ne i} S_{ik}y_k^* - S_{ii}\operatorname{sgn}(\tilde{b}_i)-\tilde{b}_i\right)^2 \\
&= 4\sum_{j\ne i}\left(\sum_{k\ne j,i} S_{jk}y_k^* + S_{jj}y_j^* - \tilde{b}_j\right) S_{ji}\operatorname{sgn}(\tilde{b}_i) \\
&\quad +4\left(\sum_{k\ne i} S_{ik}x_k^* - \tilde{b}_i\right) S_{ii}\operatorname{sgn}(\tilde{b}_i) \\
&= 4\sum_{j\ne i}\left(\sum_{k\ne j,i} S_{jk}y_k^* + S_{jj}y_j^* - \tilde{b}_j\right) S_{ji}\operatorname{sgn}(\tilde{b}_i) \\
&\quad +4\left(\sum_{k\ne i} S_{ik}x_k^*\right) S_{ii}\operatorname{sgn}(\tilde{b}_i) - 4\tilde{b}_i S_{ii}\operatorname{sgn}(\tilde{b}_i) \\
&= 4\left[\sum_{j\ne i}\left(\sum_{k\ne j,i} S_{jk}y_k^* + S_{jj}y_j^* - \tilde{b}_j\right) S_{ji}\right. \\
&\qquad \left.+\left(\sum_{k\ne i} S_{ik}x_k^*\right) S_{ii}\right]\operatorname{sgn}(\tilde{b}_i) - 4S_{ii}\left|\tilde{b}_i\right| \\
&\le 4\left[\sum_{j\ne i}\left(\sum_{k\ne j,i}\left|S_{jk}\right| + \left|S_{jj}\right| + \left|\tilde{b}_j\right|\right)\left|S_{ji}\right|\right. \\
&\quad \left.+\left(\sum_{k\ne i}|S_{ik}|\right) S_{ii}\right] - 4S_{ii}\left|\tilde{b}_i\right| \\
&= 4\left[\sum_{j\ne i}\left(\sum_{k\ne i}\left|S_{jk}\right| + \left|\tilde{b}_j\right|\right)\left|S_{ji}\right|\right. \\
&\qquad \left.+\left(\sum_{k\ne i}|S_{ik}|\right) S_{ii}\right] - 4S_{ii}\left|\tilde{b}_i\right|. \qquad (34)
\end{aligned}$$

So if $\left|\tilde{b}_i\right| > C_{0i}$, the right hand side will be negative, hence the objective function value is reduced, which contradicts the optimality of $\boldsymbol{y}^*$. □

Appendix C
Proof of Lemma 3

Let $\bar{b}_i = d(\boldsymbol{Y}\boldsymbol{b})_i$ and $\tilde{C}_{3\epsilon} = \left[(1+\tilde{C}_{1\epsilon})+(d-1)\tilde{C}_{2\epsilon}\right]d$. One should be reminded that in our analysis, we treat each entry of $\boldsymbol{Y}$ as a random variable, and $\boldsymbol{b}$ is regarded as a

constant vector so the following expectations and probabilities are on $\boldsymbol{Y}$ only. We define the following events:

$$\mathcal{F}_\epsilon: \quad \left|\Phi_{ii} - \frac{1}{\sqrt{n}}\right| \le \frac{\tilde{C}_{1\epsilon}}{\sqrt{n}}, \quad \left|\Phi_{ij}\right| \le \frac{\tilde{C}_{2\epsilon}}{\sqrt{n}}, \ \forall j \ne i, \tag{35}$$

$$\mathcal{G}_\alpha: \quad \max_i |\bar{b}_i| < d\alpha n, \tag{36}$$

and

$$\mathcal{H}_{\epsilon,\alpha}: \quad \max_i \left|\tilde{b}_i\right| < \tilde{C}_{3\epsilon}\alpha\sqrt{n}. \tag{37}$$

We first show that $\mathcal{H}_{\epsilon,\alpha}$ holds almost surely, which implies that each entry of $\tilde{\boldsymbol{b}}$ is very unlikely to be larger than the magnitude of $O(\alpha\sqrt{n})$ when n is large.

Proposition 3: Based on assumption 1, $\mathcal{P}(H_{\epsilon,\alpha}) \ge 1 - d(d-1)\exp(-\frac{n\epsilon^2}{2}) - 2d\exp(-\frac{n\alpha^2}{2})$.

Proof: Note that $\bar{b}_i = d\sum_{j=1}^n Y_{ij}b_j$ is a function of the independent variables $Y_{ij}, j = 1, 2, \cdots, n$ with a bounded difference when any of Y_{ij} flips its sign. Besides, $E_Y\left[Y_{ij}b_j\right] = b_j\, E_Y\left[Y_{ij}\right] = 0$ according to our assumption, so $E_Y\left[\bar{b}_i\right] = 0$. By McDiarmid's Inequality, we have

$$\mathcal{P}(|\bar{b}_i| > d\alpha n) \le 2\exp(-\frac{n\alpha^2}{2}), \tag{38}$$

from which we conclude that

$$\mathcal{P}(\mathcal{G}_\alpha) \ge 1 - 2d\exp(-\frac{n\alpha^2}{2}). \tag{39}$$

Whenever both $\mathcal{F}_\epsilon$ and $\mathcal{G}_\alpha$ occur, we have

$$\begin{aligned}\left|\tilde{b}_i\right| &= \left|\sum_k \Phi_{ik}\bar{b}_k\right| \le d\alpha n\sum_k |\Phi_{ik}| \\ &\le d\alpha n\left(|\Phi_{ii}| + \sum_{k\ne i}|\Phi_{ik}|\right) \\ &\le \frac{d(1+\tilde{C}_{1\epsilon}+(d-1)\tilde{C}_{2\epsilon})\alpha n}{\sqrt{n}} = \tilde{C}_{3\epsilon}\alpha\sqrt{n}.\end{aligned} \tag{40}$$

This suggests that $\mathcal{H}_{\epsilon,\alpha}$ holds. Thus

$$\begin{aligned}\mathcal{P}(\mathcal{H}_{\epsilon,\alpha}) &\ge \mathcal{P}(\mathcal{F}_\epsilon \wedge \mathcal{G}_\alpha) \ge 1 - \mathcal{P}(\neg\mathcal{F}_\epsilon) - \mathcal{P}(\neg\mathcal{G}_\alpha) \\ &\ge 1 - d(d-1)\exp(-\frac{n\epsilon^2}{2}) - 2d\exp(-\frac{n\alpha^2}{2}).\end{aligned} \tag{41}$$

□

To prove Lemma 3, we will show that if $\left|\tilde{b}_i\right| \ge C_{0i}$ and $\operatorname{sgn}\left(\tilde{b}_i\right) \ne \operatorname{sgn}\left((\boldsymbol{\Phi}\tilde{b})_i\right)$, there exists $j \ne i$ such that $\left|\tilde{b}_j\right| \ge O(\alpha\sqrt{n})$, which rarely happens.

Proof: According to Proposition 2 and Proposition 3, $\mathcal{F}_\epsilon \wedge \mathcal{H}_{\epsilon,\alpha}$ holds with probability at least $1 - 2d(d-1)\exp(-\frac{n\epsilon^2}{2}) - 2d\exp\left(-\frac{C_{0i}^2}{2\left[(d-1)(2d-1)\tilde{C}_{3\epsilon}\epsilon\right]^2}\right)$, where we set $\alpha = \frac{C_{0i}}{(d-1)(2d-1)\tilde{C}_{3\epsilon}\sqrt{n}\epsilon}$. Next we are going to show that the sign of $\tilde{b}_i$ and $(\boldsymbol{\Phi}\tilde{b})_i$ are always the same whenever $\mathcal{F}_\epsilon \wedge \mathcal{H}_{\epsilon,\alpha}$ holds. If $\operatorname{sgn}\left(\tilde{b}_i\right) \ne \operatorname{sgn}\left((\boldsymbol{\Phi}\tilde{b})_i\right)$, then as the signs of $(\boldsymbol{\Phi}\tilde{b})_i$ and $-\Phi_{ii}\tilde{b}_i$ are the same, we have

$$\sum_{k\ne i}|\Phi_{ik}|\left|\tilde{b}_k\right| \ge \left|\sum_{k\ne i}\Phi_{ik}\tilde{b}_k\right| = \left|(\boldsymbol{\Phi}\tilde{b})_i - \Phi_{ii}\tilde{b}_i\right| \ge \left|\Phi_{ii}\tilde{b}_i\right|. \tag{42}$$

This implies that there exists $j\,(j \ne i)$, such that

$$\left|\tilde{b}_j\right| \ge \frac{1}{d-1}\frac{|\Phi_{ii}|}{|\Phi_{ij}|}\left|\tilde{b}_i\right|. \tag{43}$$

As $\mathcal{F}_\epsilon$ occurs,

$$\left|\tilde{b}_j\right| \ge \frac{1}{d-1}\frac{1-\tilde{C}_{1\epsilon}}{\tilde{C}_{2\epsilon}}C_{0i}. \tag{44}$$

Since $\epsilon \le \frac{1}{2(d-1)}$, we have $(d-1)\epsilon \le \frac{1}{2}$. Thus we may set $\tilde{c} = 2$ in Proposition 2 since $\left|\frac{1}{\sqrt{1+\gamma}} - 1 + \frac{1}{2}\gamma\right| \le \tilde{c}\gamma^2$ holds when $|\gamma| < (d-1)\epsilon \le \frac{1}{2}$. In this case, $\tilde{C}_{3\epsilon} = d(1+2d(d-1)^2\epsilon^2 + \frac{(d-1)\epsilon}{2})$. Then

$$\tilde{C}_{1\epsilon} = \tilde{c}(d-1)^2\epsilon^2 \le \frac{1}{2}, \tag{45}$$

$$\tilde{C}_{2\epsilon} = \tilde{c}(d-1)^2\epsilon^2 + \frac{\epsilon}{2} \le \frac{[2(d-1)+1]\epsilon}{2} = \frac{(2d-1)\epsilon}{2}. \tag{46}$$

These facts imply that

$$\left|\tilde{b}_j\right| \ge \frac{C_{0i}}{(d-1)(2d-1)\epsilon} = \frac{C_{0i}\tilde{C}_{3\epsilon}\sqrt{n}}{(d-1)(2d-1)\tilde{C}_{3\epsilon}\sqrt{n}\epsilon}. \tag{47}$$

It contradicts with the occurrence of $\mathcal{H}_{\epsilon,\alpha}$ with $\alpha = \frac{C_{0i}}{(d-1)(2d-1)\tilde{C}_{3\epsilon}\sqrt{n}\epsilon}$. Thus

$$\begin{aligned}&\mathcal{P}\left(\operatorname{sgn}(\tilde{b}_i) = \operatorname{sgn}((\boldsymbol{\Phi}\tilde{b})_i)\right) \\ &\ge \mathcal{P}(\mathcal{F}_\epsilon \wedge \mathcal{H}_{\epsilon,\alpha}) \\ &\ge 1 - \mathcal{P}(\neg\mathcal{F}_\epsilon) - \mathcal{P}(\neg\mathcal{H}_{\epsilon,\alpha}) \\ &\ge 1 - 2d(d-1)\exp(-\frac{n\epsilon^2}{2}) - 2d\exp(-\frac{n\alpha^2}{2}) \\ &= 1 - 2d(d-1)\exp(-\frac{n\epsilon^2}{2}) \\ &\quad - 2d\exp\left(-\frac{C_{0i}^2}{2\left[(d-1)(2d-1)\tilde{C}_{3\epsilon}\epsilon\right]^2}\right).\end{aligned} \tag{48}$$

□

Appendix D
Proof of Theorem 1

Proof: According to Lemma 3, we have $\tilde{y}_i = \operatorname{sgn}\left((\boldsymbol{\Phi}\tilde{\boldsymbol{b}})_i\right) = \operatorname{sgn}(\tilde{b}_i)$ with probability at least $1 - 2d(d-1)\exp(-\frac{n\epsilon^2}{2}) - 2d\exp\left(-\frac{C_{0i}^2}{2\left[(d-1)(2d-1)\tilde{C}_{3\epsilon}\epsilon\right]^2}\right)$. So with probability at least $1 - 2d(d-1)r\exp(-\frac{n\epsilon^2}{2}) - 2d\sum_{i=1}^r\exp\left(-\frac{C_{0i}^2}{2\left[(d-1)(2d-1)\tilde{C}_{3\epsilon}\epsilon\right]^2}\right)$, the sign of first r

bits of $\tilde{\boldsymbol{b}}$ and $\boldsymbol{\Phi}\tilde{\boldsymbol{b}}$ are the same. If this holds, the difference between the objective function values with $\boldsymbol{y}^*$ and $\tilde{\boldsymbol{y}}$ is:

$$
\begin{aligned}
&\sum_{1\le i\le r}\left(\sum_{1\le k\le r,k\ne i} S_{ik}\,\mathrm{sgn}(\tilde{b}_k)+S_{ii}\,\mathrm{sgn}\left(\tilde{b}_i\right)+\sum_{k>r} S_{ik}\tilde{y}_k-\tilde{b}_i\right)^2\\
&\quad+\sum_{i>r}\left(\sum_{1\le k\le r} S_{ik}\,\mathrm{sgn}\left(\tilde{b}_k\right)+S_{ii}\tilde{y}_i+\sum_{k>r,k\ne i} S_{ik}\tilde{y}_k-\tilde{b}_i\right)^2\\
&\quad-\sum_{1\le i\le r}\left(\sum_{1\le k\le r,k\ne i} S_{ik}\,\mathrm{sgn}\left(\tilde{b}_k\right)+S_{ii}\,\mathrm{sgn}\left(\tilde{b}_i\right)\right.\\
&\qquad\qquad\left.+\sum_{k>r} S_{ik}y_k^*-\tilde{b}_i\right)^2\\
&\quad-\sum_{i>r}\left(\sum_{1\le k\le r} S_{ik}\,\mathrm{sgn}\left(\tilde{b}_k\right)+S_{ii}y_i^*+\sum_{k>r,k\ne i} S_{ik}y_k^*-\tilde{b}_i\right)^2\\
&=\sum_{1\le i\le r}\left[2\left(\sum_{1\le k\le r,k\ne i} S_{ik}\,\mathrm{sgn}(\tilde{b}_k)+S_{ii}\,\mathrm{sgn}(\tilde{b}_i)-\tilde{b}_i\right)\right.\\
&\qquad\left.+\sum_{k>r} S_{ik}(\tilde{y}_k+y_k^*)\right]\times\left(\sum_{k>r} S_{ik}(\tilde{y}_k-y_k^*)\right)\\
&\quad+\sum_{i>r}\left[2\left(\sum_{1\le k\le r} S_{ik}\,\mathrm{sgn}(\tilde{b}_k)-\tilde{b}_i\right)\right.\\
&\qquad\left.+S_{ii}\left(\tilde{y}_i+y_i^*\right)+\sum_{k>r,k\ne i} S_{ik}(\tilde{y}_k+y_k^*)\right]\\
&\quad\times\left[S_{ii}\left(\tilde{y}_i-y_i^*\right)+\sum_{k>r,k\ne i} S_{ik}(\tilde{y}_k-y_k^*)\right]\\
&=\sum_{1\le i\le r}\left[2\left(\sum_{1\le k\le r,k\ne i} S_{ik}\,\mathrm{sgn}(\tilde{b}_k)+S_{ii}\,\mathrm{sgn}(\tilde{b}_i)-\tilde{b}_i\right)\right.\\
&\qquad\left.+\sum_{k>r} S_{ik}(\tilde{y}_k+y_k^*)\right]\times\left(\sum_{k>r} S_{ik}(\tilde{y}_k-y_k^*)\right)\\
&\quad+\sum_{i>r}\left\{\left[2\left(\sum_{1\le k\le r} S_{ik}\,\mathrm{sgn}(\tilde{b}_k)-\tilde{b}_i\right)\right.\right.\\
&\qquad\left.\left.+\sum_{k>r,k\ne i} S_{ik}(\tilde{y}_k+y_k^*)\right]\left[S_{ii}\left(\tilde{y}_i-y_i^*\right)\right]\right.\\
&\quad\left.+\left[2\left(\sum_{1\le k\le r} S_{ik}\,\mathrm{sgn}(\tilde{b}_k)-\tilde{b}_i\right)+S_{ii}(\tilde{y}_i+y_i^*)\right.\right.\\
&\qquad\left.\left.+\sum_{k>r,k\ne i} S_{ik}(\tilde{y}_k+y_k^*)\right]\times\left(\sum_{k>r,k\ne i} S_{ik}(\tilde{y}_k-y_k^*)\right)\right\},
\end{aligned}
\tag{49}
$$

where we have used the fact that $\tilde{y}_i^2=\left(y_i^*\right)^2=1$. Then the above equation

$$
\begin{aligned}
&\le\sum_{1\le i\le r}\left[2\left(\sum_{1\le k\le r,k\ne i}|S_{ik}|+S_{ii}+|\tilde{b}_i|\right)+2\sum_{k>r}|S_{ik}|\right]\\
&\quad\times\left(\sum_{k>r}2|S_{ik}|\right)\\
&\quad+\sum_{i>r}\left[\left(2(\sum_{1\le k\le r}|S_{ik}|+|\tilde{b}_i|)+2\sum_{k>r,k\ne i}|S_{ik}|\right)(2S_{ii})\right.\\
&\qquad+\left(2(\sum_{1\le k\le r}|S_{ik}|+|\tilde{b}_i|)+2S_{ii}+2\sum_{k>r,k\ne i}|S_{ik}|\right)\\
&\qquad\left.\times\left(\sum_{k>r,k\ne i}2|S_{ik}|\right)\right]\\
&=4\sum_{1\le i\le r}\left(\sum_k|S_{ik}|+|\tilde{b}_i|\right)\left(\sum_{k>r}|S_{ik}|\right)\\
&\quad+4\sum_{i>r}\left[\left(\sum_k|S_{ik}|-S_{ii}+|\tilde{b}_i|\right)S_{ii}\right.\\
&\qquad\left.+\left(\sum_k|S_{ik}|+|\tilde{b}_i|\right)\left(\sum_{k>r,k\ne i}|S_{ik}|\right)\right]\\
&=4\sum_{1\le i\le r}\left(\sum_k|S_{ik}|+|\tilde{b}_i|\right)\left(\sum_{k>r}|S_{ik}|\right)\\
&\quad+4\sum_{i>r}\left[\left(\sum_k|S_{ik}|+|\tilde{b}_i|\right)\left(\sum_{k>r}|S_{ik}|\right)-S_{ii}^2\right]\\
&=4\sum_i\left(\sum_k|S_{ik}|+|\tilde{b}_i|\right)\left(\sum_{k>r}|S_{ik}|\right)-4\sum_{i>r}S_{ii}^2.
\end{aligned}
\tag{50}
$$

Thus we prove the theorem. □

REFERENCES

[1] D. P. Bertsekas, *Nonlinear Programming*, 2nd ed. Belmont, MA, USA: Athena Scientific, 1999.

[2] I. Borg and P. Groenen, "Modern multidimensional scaling: Theory and applications," *J. Edu. Meas.*, vol. 40, pp. 277–280, 2003.

[3] F. Cakir and S. Sclaroff, "Adaptive hashing for fast similarity search," in *Proc. IEEE Int. Conf. Comput. Vis.*, Dec. 2015, pp. 1044–1052.

[4] T. Dean, M. A. Ruzon, M. Segal, J. Shlens, S. Vijayanarasimhan, and J. Yagnik, "Fast, accurate detection of 100,000 object classes on a single machine," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2013, pp. 1814–1821.

[5] V. E. Liong, J. Lu, G. Wang, P. Moulin, and J. Zhou, "Deep hashing for compact binary codes learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 2475–2483.

[6] A. Friedman, "Framing pictures: The role of knowledge in automatized encoding and memory for gist," *J. Experim. Psychol., Gen.*, vol. 108, no. 3, pp. 316–355, 1979.

[7] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Trans. Math. Softw.*, vol. 3, no. 3, pp. 209–226, 1977.

[8] T. Ge, K. He, Q. Ke, and J. Sun, "Optimized product quantization for approximate nearest neighbor search," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2013, pp. 2946–2953.

[9] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proc. Int. Conf. Very Large Data Bases*, vol. 99. 1999, pp. 518–529.
[10] Y. Gong, S. Kumar, H. A. Rowley, and S. Lazebnik, "Learning binary codes for high-dimensional data using bilinear projections," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2013, pp. 484–491.
[11] Y. Gong, S. Kumar, V. Verma, and S. Lazebnik, "Angular quantization-based binary codes for fast similarity search," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1196–1204.
[12] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin, "Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 12, pp. 2916–2929, Dec. 2013.
[13] L.-K. Huang, Q. Yang, and W.-S. Zheng, "Online hashing," in *Proc. 23th Int. Joint Conf. Artif. Intell.*, 2013, pp. 1422–1428.
[14] H. Jegou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, pp. 117–128, Jan. 2011.
[15] Q.-Y. Jiang and W.-J. Li, "Scalable graph hashing with feature transformation," in *Proc. 24th Int. Joint Conf. Artif. Intell.*, 2015, pp. 2248–2254.
[16] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," M.S. thesis, Univ. of Toronto, Toronto, ON, Canada, 2009.
[17] B. Kulis and T. Darrell, "Learning to hash with binary reconstructive embeddings," in *Proc. Adv. Neural Inf. Process. Syst.*, 2009, pp. 1042–1050.
[18] C. Leng, J. Wu, J. Cheng, X. Bai, and H. Lu, "Online sketching hashing," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 2503–2511.
[19] P. Li, A. Shrivastava, J. L. Moore, and A. C. König, "Hashing algorithms for large-scale learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2011, pp. 2672–2680.
[20] G. Lin, C. Shen, and A. van den Hengel, "Supervised hashing using graph cuts and boosted decision trees," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 11, pp. 2317–2331, Nov. 2015.
[21] W. Liu, C. Mu, S. Kumar, and S.-F. Chang, "Discrete graph hashing," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 3419–3427.
[22] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang, "Supervised hashing with kernels," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2012, pp. 2074–2081.
[23] W. Liu, J. Wang, S. Kumar, and S.-F. Chang, "Hashing with graphs," in *Proc. 28th Int. Conf. Mach. Learn.*, 2011, pp. 1–8.
[24] X. Liu, C. Deng, B. Lang, D. Tao, and X. Li, "Query-adaptive reciprocal hash tables for nearest neighbor search," *IEEE Trans. Image Process.*, vol. 25, no. 2, pp. 907–919, Feb. 2016.
[25] K. B. Petersen and M. S. Pedersen, "The matrix cookbook," Tech. Univ. Denmark, Lyngby, Denmark, Tech. Rep., 2008, p. 15, vol. 7.
[26] F. Shen, W. Liu, S. Zhang, Y. Yang, and H. T. Shen, "Learning binary codes for maximum inner product search," in *Proc. IEEE Int. Conf. Comput. Vis.*, Dec. 2015, pp. 4148–4156.
[27] F. Shen, C. Shen, W. Liu, and H. T. Shen, "Supervised discrete hashing," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 37–45.
[28] F. Shen, C. Shen, Q. Shi, A. van den Hengel, Z. Tang, and H. T. Shen, "Hashing on nonlinear manifolds," *IEEE Trans. Image Process.*, vol. 24, no. 6, pp. 1839–1851, Jun. 2015.
[29] A. Shrivastava and P. Li, "Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS)," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2321–2329.
[30] C. Strecha, A. Bronstein, M. Bronstein, and P. Fua, "LDAHash: Improved matching with smaller descriptors," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 1, pp. 66–78, Jan. 2012.
[31] J. Tang, Z. Li, M. Wang, and R. Zhao, "Neighborhood discriminant hashing for large-scale image retrieval," *IEEE Trans. Image Process.*, vol. 24, no. 9, pp. 2827–2840, Sep. 2015.
[32] 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.*, vol. 30, no. 11, pp. 1958–1970, Nov. 2008.
[33] J. Wang, S. Kumar, and S.-F. Chang, "Semi-supervised hashing for large-scale search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 12, pp. 2393–2406, Dec. 2012.
[34] J. Wang, W. Liu, S. Kumar, and S.-F. Chang, "Learning to hash for indexing big data–A survey," *Proc. IEEE*, vol. 104, no. 1, pp. 34–57, Jan. 2016.
[35] Y. Weiss, R. Fergus, and A. Torralba, "Multidimensional spectral hashing," in *Proc. Eur. Conf. Comput. Vis.*, 2012, pp. 340–353.
[36] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Proc. Adv. Neural Inf. Process. Syst.*, 2009, pp. 1753–1760.
[37] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba, "Sun database: Large-scale scene recognition from abbey to zoo," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2010, pp. 3485–3492.
[38] F. Yu, S. Kumar, Y. Gong, and S.-F. Chang, "Circulant binary embedding," in *Proc. 31st Int. Conf. Mach. Learn.*, 2014, pp. 946–954.
[39] D. Zhang, J. Wang, D. Cai, and J. Lu, "Self-taught hashing for fast similarity search," in *Proc. 33rd Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, 2010, pp. 18–25.

Yameng Huang received the B.E. degree in computer science and technology from the School of Electronics Engineering and Computer Science, Peking University, Beijing, China, in 2015, where he is currently pursuing the M.S. degree. His research interests include computer vision, image processing, and machine learning.

Zhouchen Lin (M'00–SM'08) received the Ph.D. degree in applied mathematics from Peking University in 2000. He is currently a Professor with the Key Laboratory of Machine Perception, School of Electronics Engineering and Computer Science, Peking University. His research areas include computer vision, image processing, machine learning, pattern recognition, and numerical optimization. He is an area chair of CVPR 2014/2016, ICCV 2015, and NIPS 2015, and a senior program committee member of AAAI 2016/2017/2018 and IJCAI 2016/2018. He is an associate editor of the IEEE *Transactions on Pattern Analysis and Machine Intelligence* and the *International Journal of Computer Vision*. He is an IAPR Fellow.