Distributed Graph Hashing

Shengnan Wang, Chunguang Li, *Senior Member, IEEE*, and Hui-Liang Shen

***Abstract*—Recently, hashing-based approximate nearest neighbors search has attracted considerable attention, especially in big data applications, due to its low computation cost and fast retrieval speed. In the literature, most of the existing hashing algorithms are centralized. However, in many large-scale applications, the data are often stored or collected in a distributed manner. In this situation, the centralized hashing methods are not suitable for learning hash functions. In this paper, we consider the distributed learning to hash problem. We propose a novel distributed graph hashing model for learning efficient hash functions based on the data distributed across multiple agents over network. The graph hashing model involves a graph matrix, which contains the similarity information in the original space. We show that the graph matrix in the proposed distributed hashing model can be decomposed into multiple local graph matrices, and each local graph matrix can be constructed by a specific agent independently, with moderate communication and computation cost. Then, the whole objective function of the distributed hashing model can be represented by the sum of local objective functions of multiple agents, and the hashing problem can be formulated as a nonconvex constrained distributed optimization problem. For tractability, we transform the nonconvex constrained distributed optimization problem into an equivalent bi-convex distributed optimization problem. Then we propose two algorithms based on the idea of alternating direction method of multipliers to solve this problem in a distributed manner. We show that the proposed two algorithms have moderate communication and computational complexities, and both of them are scalable. Experiments on benchmark datasets are given to demonstrate the effectiveness of the proposed methods.**

***Index Terms*—Alternating direction method of multipliers (ADMMs), distributed hashing, graph hashing, large-scale image retrieval, learning to hash (LH).**

I. Introduction

HASHING-BASED approximate nearest neighbor search has attracted considerable attention in many areas, including information retrieval, image search, machine learning, and computer vision, etc. [1]–[13]. The main idea of hashing is to encode high-dimensional data points into compact binary codes, and the binary codes should be similarity preserving. In other words, the hashing technique maps similar data points to adjacent binary hash codes. Through hashing, nearest neighbor search can be accomplished with a constant time complexity, which enables large efficiency gains in both storage and computation speed.

Existing hashing methods can be mainly divided into two categories: 1) data-independent and 2) data-dependent schemes. Data-independent hashing methods do not use any data information for obtaining hash functions. Locality sensitive hashing [14] is one of the most widely known data-independent hashing methods, and it generates hash functions based on random projections. Recently, more works focus on data-dependent hashing, for it can effectively and efficiently map massive data points to very short compact binary codes. Representative approaches include spectral hashing [4], spherical hashing [21], iterative quantization (ITQ) [3], structure sensitive hashing [51], adaptive binary quantization (ABQ) [17], and a recently proposed fast optimization method DPLM [11] which can solve general binary code learning problems. Different from data-independent hashing, data-dependent hashing methods learn binary codes and hash functions using data, so they are also called learning to hash (LH) methods [25].

LH methods mainly have two types: 1) unsupervised hashing [3], [4], [15], [16], [18]–[20], [27], [28], [51] and 2) supervised hashing [22], [24], [26], [32]–[35], [38]. The main difference between the two types is whether the label information of the data is available for learning hash functions. Generally, supervised hashing can better preserve the semantic similarity in the original space and is demonstrated to achieve better accuracy, compared with unsupervised hashing. Though the supervised hashing has promising performance, this scheme has a fundamental drawback that it requires the supervised labels. In many real-world applications, it is usually expensive or even impossible to get the semantic labels, so in these cases only unsupervised hashing can be performed. Hence, in this paper, we focus on unsupervised hashing.

Among the unsupervised hashing methods, graph hashing is expected to achieve better performance than other unsupervised hashing methods if the learning algorithms are effective enough, for graph hashing learns hash codes and hash functions by directly exploiting the similarity (neighborhood structure), which typically reflects the pairwise relationship between two data points. The objective of graph hashing exactly matches the goal of similarity-preserving hashing. In addition, graph hashing usually requires the bits of the binary codes to be uncorrelated and balanced [4], [29], which further guarantees the efficiency of hashing. The existing graph hashing methods include spectral hashing, anchor graph hashing [8], discrete graph hashing [29], ordinal constrained

Manuscript received September 8, 2017; revised December 28, 2017 and March 7, 2018; accepted March 14, 2018. Date of publication April 5, 2018; date of current version March 5, 2019. This work was supported in part by the National Natural Science Foundation of China under Grant 61571392, Grant 61471320, and Grant 61631003, and in part by the National Program for Special Support of Eminent Professionals. This paper was recommended by Associate Editor D. Tao. *(Corresponding author: Chunguang Li.)*

The authors are with the College of Information Science and Electronic Engineering and the Zhejiang Provincial Key Laboratory of Information Processing, Communication and Networking, Zhejiang University, Hangzhou 310027, China (e-mail: cgli@zju.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/TCYB.2018.2816791

hashing [30], scalable graph hashing (SGH) [31], and so on. In the literature, most of the graph hashing methods have superior accuracy compared with other unsupervised hashing methods, especially for long-bit codes.

All the hashing methods mentioned above are centralized. In other words, these methods are only suitable for leaning hash functions by a single machine. However, in real-world applications, we need to deal with many big-data problems, such as large-scale image processing [36], [37], large-scale information retrieval [39]. These problems are often beyond the capacity of a single machine (computer). In this situation, the data is usually partitioned and stored in multiple machines. Furthermore, in the applications of search engine, the data is usually collected by many agents and the whole data should be utilized for indexing. Thus, we need to devise distributed algorithms to solve these problems. The distributed algorithm has been widely studied in a large variety of areas, including signal processing [40]–[43], machine learning [44]–[46], and automatic control [47]–[49]. In the distributed setting, each agent can only exchange information with its neighbors, and the aim is to cooperatively solve a large-scale problem with a moderate communication cost. It will benefit a lot if such a distributed manner can be adopted for learning hash functions based on the data stored/collected by multiple machines. However, as we know, such a critical issue is rarely considered in the literature. The most related methods we know are [50] and [52]. Leng *et al.* [50] gave a distributed hashing method named DisH by decomposing a centralized hashing problem into multiple subproblems. DisH obtains hash codes and hash functions by learning a dictionary matrix. Liu *et al.* [52] proposed a distributed ABQ (DABQ) by extending the ABQ method into a distributed learning framework, which speeds up the training of ABQ. However, these methods do not fully utilize the similarity in the original space. In addition, these methods do not constrain the bits of the binary codes to be uncorrelated and balanced, while bits uncorrelation and balance are two key features of compact binary code learning [4].

In this paper, to fully utilize the similarity among the data samples, we propose a novel graph hashing model for learning efficient hash functions in a distributed manner. Inspired by [8], we construct an anchor point-based graph matrix for modeling the distributed graph hashing. We show that the graph matrix in the proposed distributed hashing model can be decomposed into multiple local graph matrices, and each local graph matrix can be constructed by a specific agent independently, with moderate communication and computation cost, which satisfies the distributed setting. Using this graph matrix, we cast our graph hashing model. We add the widely used bits uncorrelation and balance constraints to make the learned binary codes and hash functions more efficient. In order to devise distributed algorithms, we decompose the objective function of this graph hashing model into a set of local objective functions. Then the whole problem can be successfully formulated as a constrained distributed optimization problem. The constrained optimization problem is a nonconvex optimization problem. For tractability, we reformulate the problem as an equivalent bi-convex optimization problem. Then we propose two distributed algorithms, respectively, named sequential distributed hashing (SDH) and parallel distributed hashing (PDH) based on the alternating direction method of multipliers (ADMMs) [46] to solve this problem. Both of the two algorithms perform well and meanwhile each of them has its own advantages. PDH is more time efficient, while SDH consumes less resource.

The rest of this paper is organized as follows. In Section II, we introduce some necessary notations and preliminaries. In Section III, we propose a distributed graph hashing model and give the problem formulation. We propose two distributed algorithms to solve the problem in Section IV. The communication and computational complexity analyses are given in Section V. Numerical simulations on large-scale benchmark datasets are provided in Section VI. Finally, we draw the conclusion in Section VII.

II. Preliminaries

A. Notation

In this paper, a lowercase letter is used to denote a column vector and a capital letter is used to denote a matrix. For a vector x, we use x^{T} to denote the transpose of x. We use $\|x\|$ to denote the l_2-norm of x. For a matrix X, $X_{i,j}$ denotes the entry in the ith row and jth column of X, $\|X\|_F$ denotes the Frobenius norm, and $\mathrm{tr}(X)$ denotes the trace of X. We use $\mathrm{sign}(\cdot)$ to denote the element-wise sign function. We use I to denote an identity matrix.

B. Graph Hashing

Let $X = [x_1, x_2, \ldots, x_n] \in \mathbb{R}^{d\times n}$ denote the dataset of n samples, where d is the dimensionality of the data. Without loss of generality, the data points are assumed to be zero centered, i.e., $\sum_{i=1}^{n} x_i = 0$. Hashing is to encode each high-dimensional data point x_i into a compact binary code $y_i \in \{-1, 1\}^c$, where c is the code length. Generally, we will learn c binary hash functions $\{h_k(\cdot)|k = 1, 2, \ldots, c\}$, or equivalently, a c-dimensional hash function $h(\cdot) = [h_1(\cdot), h_2(\cdot), \ldots, h_c(\cdot)]^{\mathrm{T}}$, so that the binary code of a query x can be easily computed as $y = [h_1(x), h_2(x), \ldots, h_c(x)]^{\mathrm{T}}$. Graph hashing formulates the binary code learning problem as follows:

$$\begin{aligned} \min_{\{y_i\}} \ & \sum_{i,j=1}^{n} W_{i,j}\|y_i - y_j\|^2 \\ \text{s.t.} \ & y_i \in \{-1, 1\}^c \\ & \sum_{i=1}^{n} y_i = 0 \\ & \frac{1}{n}\sum_{i}^{n} y_i y_i^{\mathrm{T}} = I \end{aligned} \tag{1}$$

where $W = [W_{i,j}]_{n\times n}$ is called the graph matrix and $W_{i,j} = \exp(-\|x_i - x_j\|^2/\epsilon)$, denoting the Euclidean similarity between x_i and x_j. Here, parameter $\epsilon > 0$ defines the distance in $\mathbb{R}^d$ which corresponds to similar items. The last two constraints force the binary codes to be balanced and uncorrelated, respectively. From the objective function we see that graph hashing tends to assign adjacent binary codes for similar data points, which matches the goal of similarity-preserving hashing.

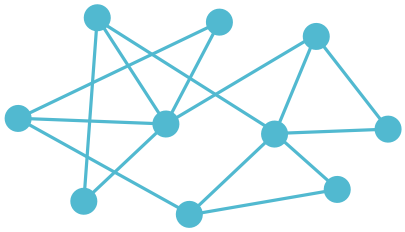


Fig. 1. Randomly generated network with 10 agents.

C. Network Model

In this paper, we consider that the data is distributed across m agents (each agent can be seen as a single machine), and the m agents together constitute a network (e.g., Fig. 1). The topology of the network is represented by an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} := \{1, 2, \ldots, m\}$ denotes the set of vertices, and $\mathcal{E} \subset \{\{l, j\} | l, j \in \mathcal{V}, l \neq j\}$ denotes the set of edges. Each vertex of the graph is referred to as an agent. The edge $\{l, j\} \in \mathcal{E}$ indicates that agents l and j are neighbors which can directly exchange information with each other through an undirected link. We use N_l to denote the set of the neighbors of agent l (excluding itself), and $|N_l|$ to denote the degree of agent l. We assume the whole network is strongly connected so that any two agents can communicate directly or indirectly with each other.

The network model can describe various networks. In different networks, the agents represent different machines. For example, in the wireless sensor network, each agent represents a sensor node; in the small cell networks, each agent represents a base station; in the camera network, each agent represents a camera; in the computer network, each agent represents a computer.

III. Formulation of Distributed Graph Hashing

In this section, we propose a distributed graph hashing model to learn hash functions based on the data distributed across m agents over a connected network (e.g., Fig. 1). For each $l \in \mathcal{V}$, let $X^l = [x_1^l, x_2^l, \ldots, x_{n_l}^l] \in \mathbb{R}^{d \times n_l}$ denote the local dataset of agent l, whose columns are the data points in this agent. Then the whole data set X is a concatenation of local datasets, i.e., $X = [X^1, X^2, \ldots, X^m]$.

A. Distributed Graph Hashing Model

Since the data points are distributed across multiple agents, it is difficult to construct a neighborhood graph matrix, such as the matrix $W = [W_{i,j}]_{n \times n}$ in (1), for modeling the distributed graph hashing, due to unacceptable communication and computation cost. Inspired by [8], we choose to construct an anchor point-based graph matrix instead of the neighborhood graph matrix, and we will show that such a graph matrix can be computed and stored in a distributed manner with moderate communication and computation cost. To achieve this, we need to select $q\,(q \ll n)$ representative anchor points $\{a_j\}_{j=1}^q$ from the whole data points distributed across multiple agents. A simple way is that each agent l randomly chooses q_l data points from the local dataset as local anchor points, and broadcasts them beforehand. The total number of all the anchor points is $q = \sum_{l=1}^m q_l$. For better performance, to make the anchor points be more representative of the whole data points, a more efficient way is using the method of distributed vector quantization [54], [55] to generate the anchor points. The distributed vector quantization method generates more anchor points in the high-density region and less in the low-density region of the whole dataset, which is more reasonable, though it consumes more computing resources. When the anchor points are obtained, we use them to construct a graph matrix $\tilde{W} \in \mathbb{R}^{n \times q}$ measuring the similarity between the training data points and the anchor points, where $\tilde{W}_{i,j} = \exp(-\|x_i - a_j\|^2/\epsilon)$. The graph matrix $\tilde{W}$ can be represented as a concatenation of local graph matrices, i.e., $\tilde{W} = [\tilde{W}^1; \tilde{W}^2; \cdots; \tilde{W}^m]$, where $\tilde{W}^l \in \mathbb{R}^{n_l \times q}$ is the local graph matrix of agent l, and

$$\tilde{W}_{i,j}^l = \exp(-\|x_i^l - a_j\|^2/\epsilon). \tag{2}$$

The local graph matrix $\tilde{W}^l$ can be independently computed by agent l, and the time complexity for each agent l to compute $\tilde{W}^l$ is moderate, which satisfies the requirement of the distributed setting.

After constructing the graph matrix, we train binary hash codes by solving the following problem:

$$\begin{aligned} \min_{\{y_i, b_j\}} \ & \sum_{i=1}^n \sum_{j=1}^q \tilde{W}_{i,j} \|y_i - b_j\|^2 \\ \text{s.t.} \ & y_i \in \{-1, 1\}^c, b_j \in \{-1, 1\}^c \\ & \sum_{i=1}^n y_i = 0 \\ & \frac{1}{n} \sum_{i=1}^n y_i y_i^{\mathrm{T}} = I \end{aligned} \tag{3}$$

where y_i is the binary code of the data point x_i, and b_j is the binary code of the anchor point a_j. Note that $\|y_i\|^2 = \|b_j\|^2 = c$ for all i, j, then the objective function of problem (3) can be replaced by $\sum_{i=1}^n \sum_{j=1}^q -\tilde{W}_{i,j} y_i^T b_j$. Let $Y = [y_1, y_2, \ldots, y_n] \in \mathbb{R}^{c \times n}$ denote the code matrix of the whole data, and let $B = [b_1, b_2, \ldots, b_q] \in \mathbb{R}^{c \times q}$ denote the code matrix of the anchor points. Then the above problem can be rewritten in a compact matrix form as

$$\begin{aligned} \min_{Y,B} \ & -\mathrm{tr}\big(Y \tilde{W} B^{\mathrm{T}}\big) \\ \text{s.t.} \ & Y \in \{-1, 1\}^{c \times n}, B \in \{-1, 1\}^{c \times q} \\ & Y\mathbf{1} = 0 \\ & YY^{\mathrm{T}} = nI. \end{aligned} \tag{4}$$

For each agent l, we use $Y^l := [y_1^l, y_2^l, \ldots, y_{n_l}^l] \in \mathbb{R}^{c \times n_l}$ to denote the local code matrix, whose columns are the binary codes of the local data points in this agent. Then the code matrix Y can be represented as a concatenation of local code matrices, i.e., $Y = [Y^1, Y^2, \ldots, Y^m]$. Note that

$$Y\tilde{W} = \sum_{l=1}^m Y^l \tilde{W}^l$$

then the objective in (4) can be decomposed into several local objectives, and problem (4) can be reformulated as

$$\begin{aligned}\min_{\{Y^l\},B}\ & \sum_{l=1}^{m} -\mathrm{tr}\left(Y^l\tilde{W}^l B^{\mathrm{T}}\right)\\ \text{s.t.}\ & Y^l \in \{-1,1\}^{c\times n_l}, B\in\{-1,1\}^{c\times q}\\ & Y\mathbf{1}=0\\ & YY^{\mathrm{T}}=nI.\end{aligned} \tag{5}$$

B. Hash Function

Since the dimension of the variables in (5) is very large, it is difficult to directly learn binary codes Y by optimizing (5). In addition, even problem (5) is solved, it is still unable to efficiently encode into a novel query into binary code. Thus, we aim to learn a c-dimensional hash function $h(\cdot)$. Then the binary codes of the training data and a novel query can be easily computed using the hash function. We define the hash function as follows:

$$h(x)=\mathrm{sign}(P^T K(x)) \tag{6}$$

where $K(x)$ is a q-dimensional column vector defined as $K(x)=[\phi(x;a_1)-\nu_1,\ldots,\phi(x;a_q)-\nu_q]^{\mathrm{T}}$. Here, $\phi(x;a)=\exp(-\|x-a\|^2/\sigma)$ is an RBF (Gaussian) function, $\{a_j\}_{j=1}^q$ are the q anchor points mentioned above, $\{\nu_j\}_{j=1}^q$ are biases, and σ is the kernel width. The biases $\{\nu_j\}$ are typically set as $\nu_j=\sum_{i=1}^n \phi(x_i;a_j)/n$, whose effect is to make the mapped data $K(x)$ zero centered, i.e., $\sum_{i=1}^n K(x_i)=0$. The projection matrix $P=[p_1,p_2,\ldots,p_c]\in\mathbb{R}^{q\times c}$ projects the mapped data $K(x)$ onto the low dimensional space, and p_i is assumed to have unit norm, for all $i=1,2,\ldots c$. Here, c can be any positive integer, which means that the hash function can generate hash codes with any length.

To compute ν_j, each agent does not need to explicitly get all the information of $\{\phi(x_i;a_j)\}_{i=1}^n$. Note that

$$\nu_j=\sum_{i=1}^{n}\phi\left(x_i;a_j\right)/n=\sum_{l=1}^{m}\sum_{i=1}^{n_l}\phi\left(x_i^l;a_j\right)/\sum_{l=1}^{m}n_l$$

so the agents only need to broadcast the quantity $\varphi_j^l=\sum_{i=1}^{n_l}\phi(x_i^l,a_j)$ and the number of local data points n_l to obtain ν_j. Both φ_j^l and n_l can be easily computed or obtained by agent l independently. Moreover, if the data size n is known in advance, then the network only needs to broadcast $\{\varphi_j^l\}$. Similar formulations as (6) are widely used as the kernel hash functions in the literature, such as SGH [31], KSH [32]. Such hash functions are demonstrated to have better performance than linear hash functions, especially when the data is linearly inseparable [32].

Using the hash function, the binary code of a training data point or a query x can be easily computed by $y=h(x)=\mathrm{sign}(P^T K(x))$. So we have $Y^l=\mathrm{sign}(P^T K(X^l))$, and similarly $B=\mathrm{sign}(P^T K(A))$, where $K(X^l)=[K(x_1^l),K(x_2^l),\ldots,K(x_{n_l}^l)]\in\mathbb{R}^{q\times n_l}$ is the RBF mapping of all the training data in X^l, and $K(A)=[K(a_1),K(a_2),\ldots,K(a_q)]\in\mathbb{R}^{q\times q}$ is the RBF mapping of all the anchor points. By substituting $h(\cdot)$ into (5), the whole problem can be formulated as the following distributed optimization problem with respect to P:

$$\begin{aligned}\min_{P}\ & \sum_{l=1}^{m} -\mathrm{tr}\left(\mathrm{sign}\left(P^T K\left(X^l\right)\right)\tilde{W}^l\mathrm{sign}\left(P^T K(A)\right)^{\mathrm{T}}\right)\\ \text{s.t.}\ & \mathrm{sign}\left(P^T K(X)\right)*\mathbf{1}=0\\ & \mathrm{sign}\left(P^T K(X)\right)*\mathrm{sign}\left(P^T K(X)\right)^{\mathrm{T}}=nI\end{aligned} \tag{7}$$

where $K(X)=[K(X^1),K(X^2),\ldots,K(X^m)]\in\mathbb{R}^{q\times n}$.

C. Problem Relaxation

It is difficult to directly solve problem (7), since the objective function is nondifferentiable, and the constraints make the problem NP hard [4]. A tractable solution is to relax the problem, including both the objective function and the constraints.

Wang *et al.* [22] proved that the binary codes satisfying the balance constraint is equivalent to the hash function with maximum variance on data X. However, the variance of the hash function is difficult to compute, for the existence of the nondifferentiable $\mathrm{sign}(\cdot)$ function. According to [22], the maximum variance of a hash function is lower bounded by the scaled variance of the projected data. So we choose to directly maximize the variance of the projected data $P^T K(x)$. Since $K(x)$ is zero centered, i.e., $E[P^T K(x)]=\mathbf{0}$, the variance of the projected data can be expressed as

$$\mathrm{var}\left[P^T K(x)\right]=\frac{1}{n}\mathrm{tr}\left[P^T K(X)K(X)^{\mathrm{T}}P\right]. \tag{8}$$

Applying the same relaxation method in [22], problem (7) can be relaxed as

$$\begin{aligned}\min_{P}\ & \sum_{l=1}^{m} -\mathrm{tr}\left[\left(P^T K(X^l)\right)\tilde{W}^l\left(P^T K(A)\right)^{\mathrm{T}}\right]\\ & -\frac{\eta}{n}\mathrm{tr}\left[P^T K(X)K(X)^T P\right]\\ \text{s.t.}\ & P^T P=I\end{aligned} \tag{9}$$

where η is a positive scalar weighting the variance-based regularization term, and the bits uncorrelation constraint is relaxed as the projection matrix P to be orthogonal. The detailed relaxation process and related proof can be found in [22]. Note that

$$K(X)K(X)^{\mathrm{T}}=\sum_{l=1}^{m}K\left(X^l\right)K\left(X^l\right)^{\mathrm{T}}$$

so the second term of the objective function in (9) can be decomposed as

$$\mathrm{tr}\left[P^T K(X)K(X)^T P\right]=\sum_{l=1}^{m}\mathrm{tr}\left[P^T K\left(X^l\right)K\left(X^l\right)^T P\right].$$

Then the objective function in (9) can be expressed as the sum of local objective functions

$$J(P)=\sum_{l=1}^{m}\mathrm{tr}\left(P^T S^l P\right)$$

with

$$S^l=-K\left(X^l\right)\tilde{W}^l K(A)^{\mathrm{T}}-\eta' K\left(X^l\right)K\left(X^l\right)^{\mathrm{T}} \tag{10}$$

where $\eta' = \eta/n$. Here, S^l is only known by agent l. After problem relaxation, our aim now is to solve the following problem:

$$\min_P \sum_{l=1}^{m} \mathrm{tr}\left(P^T S^l P\right) \quad \text{s.t. } P^T P = I. \tag{11}$$

Note that S^l is not necessarily a positive semidefinite matrix, so the local objective functions in (11) are not necessarily convex with respect to P. Moreover, due to the nonlinear equality constraint, problem (11) is a nonconvex optimization problem, so that the conventional distributed gradient methods are not suitable for solving this problem. In next section, we will show that problem (11) can be transformed into an equivalent bi-convex optimization problem. Then we can devise distributed ADMM algorithms to solve this problem.

IV. Distributed Algorithm

ADMM is not only suitable for solving convex optimization problems, but also suitable for solving bi-convex optimization problems having convex (or bi-convex) objective functions and bi-affine constraints [46]. In this section, we show that the nonconvex optimization problem (11) can be rewritten as equivalent bi-convex problems, then we propose two distributed ADMM algorithms for solving this problem.

A. SDH: Sequential Distributed Hash Algorithm

Before giving the algorithm, we make some uninfluential but useful modifications to problem (11). Let $M^l = (S^l + S^{l\mathrm{T}})/2, \forall l \in \mathcal{V}$, so $\{M^l\}$ are all real symmetric matrices. Since $\mathrm{tr}(P^T S^l P) = \mathrm{tr}(P^T S^{l^T} P)$, for all l, then problem (11) is equivalent to the following problem:

$$\min_P \sum_{l=1}^{m} \mathrm{tr}\left(P^T M^l P\right) \quad \text{s.t. } P^T P = I. \tag{12}$$

Let μ_l be the smallest eigenvalue of M^l. Since $\{M^l\}$ are all real symmetric matrices, $\{\mu_l\}$ are all real numbers. Let $Q^l = M^l$ if $\mu_l \geq 0$, and $Q^l = M^l - \mu_l I$ if $\mu_l < 0$, for all l, so $\{Q^l\}$ are all positive semidefinite and real symmetric matrices. It is easy to see that optimizing problem (12) is equivalent to optimizing

$$\min_P \sum_{t=1}^{m} \mathrm{tr}\left(P^T Q^l P\right) \quad \text{s.t. } P^T P = I. \tag{13}$$

Since $\{Q^l\}$ are all positive semidefinite matrices, the local objective functions in (13) are all convex with respect to the optimized variable P. Note that the network is strongly connected, then problem (13) is completely equivalent to the following bi-convex optimization problem:

$$\min \sum_{l=1}^{m} \mathrm{tr}\left(P_l{}^T Q^l P_l\right) \quad \text{s.t. } P_l = P_t, \quad P_l^T P_t = I, \quad \forall l \in \mathcal{V}, t \in N_l \tag{14}$$

where P_l are local variables, and $P_l = P_t$ are consensus constraints, for all $l \in \mathcal{V}, t \in N_l$. The augmented Lagrangian function of (14) is

$$\begin{aligned} &L_\rho\left(\{P_l\}, \{\Lambda_{l,t}\}, \{\Gamma_{l,t}\}\right) \\ &\quad = \sum_{l=1}^{m} \mathrm{tr}\left(P_l{}^T Q^l P_l\right) + \sum_{l=1}^{m}\sum_{t\in N_l} \mathrm{tr}\left(\Lambda_{l,t}^{\mathrm{T}}(P_l - P_t)\right) \\ &\qquad + \sum_{l=1}^{m}\sum_{t\in N_l} \frac{\rho}{2}\|P_l - P_t\|_F^2 + \sum_{l=1}^{m}\sum_{t\in N_l} \mathrm{tr}\left(\Gamma_{l,t}^{\mathrm{T}}\left(P_l^T P_t - I\right)\right) \\ &\qquad + \sum_{l=1}^{m}\sum_{t\in N_l} \frac{\rho}{2}\|P_l^T P_t - I\|_F^2 \end{aligned} \tag{15}$$

where $\Lambda_{l,t}$ are the Lagrange multipliers corresponding to the constraints $P_l = P_t$, $\Gamma_{l,t}$ are the Lagrange multipliers corresponding to the constraints $P_l^T P_t = I$, for all $l \in \mathcal{V}, t \in N_l$, and $\rho > 0$ is a penalty parameter. ADMM minimizes L_ρ by using the following updates:

$$\begin{aligned} P_1^{k+1} &= \arg\min_{P_1} L_\rho\left(P_1, P_2^k, \ldots, P_m^k, \left\{\Lambda_{l,t}^k\right\}, \left\{\Gamma_{l,t}^k\right\}\right) \\ P_2^{k+1} &= \arg\min_{P_2} L_\rho\left(P_1^{k+1}, P_2, P_3^k, \ldots, P_m^k, \{\Lambda_{l,t}^k\}, \{\Gamma_{l,t}^k\}\right) \\ &\vdots \\ P_m^{k+1} &= \arg\min_{P_m} L_\rho\left(P_1^{k+1}, \ldots, P_{m-1}^{k+1}, P_m, \{\Lambda_{l,t}^k\}, \{\Gamma_{l,t}^k\}\right) \\ \Lambda_{l,t}^{k+1} &= \Lambda_{l,t}^k + \rho\left(P_l^{k+1} - P_t^{k+1}\right), \quad \forall l \in \mathcal{V}, t \in N_l \\ \Gamma_{l,t}^{k+1} &= \Gamma_{l,t}^k + \rho\left(\left(P_l^{k+1}\right)^T P_t^{k+1} - I\right), \quad \forall l \in \mathcal{V}, t \in N_l \end{aligned} \tag{16}$$

where $k \geq 0$ is the iteration step. For convenience, define the sets

$$\mathcal{W}_l = \{t | t \in N_l, t < l\}$$

and

$$\mathcal{U}_l = \{t | t \in N_l, t > l\}.$$

Neglecting the irrelevant terms, to obtain P_l^{k+1}, agent l needs to solve the following subproblem:

$$\begin{aligned} \min_{P_l}\ & \mathrm{tr}\left(P_l^{\mathrm{T}} Q^l P_l\right) + \sum_{t\in N_l} \mathrm{tr}\left(\left(\Lambda_{l,t}^k\right)^{\mathrm{T}} P_l\right) - \sum_{t\in N_l} \mathrm{tr}\left(\left(\Lambda_{t,l}^k\right)^T P_l\right) \\ &+ \sum_{t\in\mathcal{W}_l} \mathrm{tr}\left(\left(\Gamma_{l,t}^k\right)^T P_l^T P_t^{k+1}\right) + \sum_{t\in\mathcal{U}_l} \mathrm{tr}\left(\left(\Gamma_{l,t}^k\right)^T P_l^T P_t^k\right) \\ &+ \sum_{t\in\mathcal{W}_l} \mathrm{tr}\left(\left(\Gamma_{t,l}^k\right)^{\mathrm{T}}\left(P_t^{k+1}\right)^T P_l\right) + \sum_{t\in\mathcal{U}_l} \mathrm{tr}\left(\left(\Gamma_{t,l}^k\right)^{\mathrm{T}}\left(P_t^k\right)^T P_l\right) \\ &+ \rho \sum_{t\in\mathcal{W}_l} \left\|P_l - P_t^{k+1}\right\|_F^2 + \rho \sum_{t\in\mathcal{U}_l} \left\|P_l - P_t^k\right\|_F^2 \\ &+ \rho \sum_{t\in\mathcal{W}_l} \left\|P_l^{\mathrm{T}} P_t^{k+1} - I\right\|_F^2 + \rho \sum_{t\in\mathcal{U}_l} \left\|P_l^{\mathrm{T}} P_t^k - I\right\|_F^2. \end{aligned} \tag{17}$$

Though the above problem seems complicate, it is a convex optimization problem. Taking the derivative of (17) with

respect to P_l, and setting the derivative to zero, we get a closed-form solution $P_l^{k+1} = (C_l^k)^{-1} E_l^k$, where

$$C_l^k = 2Q^l + 2\rho|N_l|I + 2\rho \sum_{t\in\mathcal{W}_l} P_t^{k+1}\left(P_t^{k+1}\right)^{\mathrm{T}} + 2\rho \sum_{t\in\mathcal{U}_l} P_t^k\left(P_t^k\right)^{\mathrm{T}} \tag{18}$$

and

$$E_l^k = -\sum_{t\in N_l}\Lambda_{l,t}^k + \sum_{t\in N_l}\Lambda_{t,l}^k + 4\rho\sum_{t\in\mathcal{W}_l} P_t^{k+1} + 4\rho\sum_{t\in\mathcal{U}_l} P_t^k - \sum_{t\in\mathcal{W}_l} P_t^{k+1}\left(\left(\Gamma_{l,t}^k\right)^{\mathrm{T}} + \Gamma_{t,l}^k\right) - \sum_{t\in\mathcal{U}_l} P_t^k\left(\left(\Gamma_{l,t}^k\right)^{\mathrm{T}} + \Gamma_{t,l}^k\right). \tag{19}$$

The matrix C_l^k is positive definite and invertible, since $\rho > 0$ and Q^l, $P_t^{k+1}(P_t^{k+1})^{\mathrm{T}}$, $P_t^k(P_t^k)^{\mathrm{T}}$ are all positive semidefinite.

The update iteration can be greatly simplified. Let all the Lagrange multipliers be initialized to zeros. By mathematical induction, we have that $(\Gamma_{l,t}^k)^{\mathrm{T}} = \Gamma_{t,l}^k$, for all $l \in \mathcal{V}, t \in N_l$. Define $\Lambda_l = \sum_{t\in N_l}\Lambda_{l,t} - \sum_{t\in N_l}\Lambda_{t,l}$, for all $l \in \mathcal{V}$. Then (19) can be simplified as

$$E_l^k = -\Lambda_l^k - 2\sum_{t\in\mathcal{W}_l} P_t^{k+1}\Gamma_{t,l}^k - 2\sum_{t\in\mathcal{U}_l} P_t^k\Gamma_{t,l}^k + 4\rho\sum_{t\in\mathcal{W}_l} P_t^{k+1} + 4\rho\sum_{t\in\mathcal{U}_l} P_t^k \tag{20}$$

and the updates in (16) can be simplified as

$$P_l^{k+1} = \left(C_l^k\right)^{-1} E_l^k \tag{21a}$$

$$\Lambda_l^{k+1} = \Lambda_l^k + 2\rho\sum_{t\in N_l}\left(P_l^{k+1} - P_t^{k+1}\right) \tag{21b}$$

$$\Gamma_{t,l}^{k+1} = \Gamma_{t,l}^k + \rho\left(\left(P_t^{k+1}\right)^T P_l^{k+1} - I\right) \tag{21c}$$

for all $l \in \mathcal{V}$, and $t \in N_l$. The update (21a), (21b), (21c) can be independently finished by agent l with moderate communication with its neighbors, which satisfies the distributed setting. We summarize the whole process in Algorithm 1. In practice, we can simply initialize P_l by random Gaussian matrix for each $l \in \mathcal{V}$.

Algorithm 1 SDH Algorithm

Initialization Each agent l obtains q_l anchor points and broadcasts them. Then each agent l computes and broadcasts $\{\varphi_j^l\}_{j=1}^q, n_l$. Set the penalty parameter ρ appropriately. Set $\Lambda_l^0 = 0$ and $\Gamma_{t,l}^0 = 0$, $\forall t \in N_l$. Initialize the starting point P_l^0 by a random Gaussian matrix for each $l \in \mathcal{V}$. Set $k = 0$.

Each agent l computes $\tilde{W}^l$ according to (2).
Each agent l constructs $K(X^l)$ and $K(A)$.
Each agent l computes S^l according to (10).
Each agent l computes Q^l.
repeat
  for $l = 1, 2, \cdots, m$ [in order] **do**
    Compute C_l^k and E_l^k using (18) and (20).
    Compute P_l^{k+1} using (21a).
    Transmit P_l^{k+1} to the neighbors in agent l.
  end for
  for all $l = 1, 2, \cdots, m$ [in parallel] **do**
    Compute Λ_l^{k+1} by (21b).
    Compute $\Gamma_{t,l}^{k+1}$ by (21c), $\forall t \in N_l$.
  end for
  $k = k + 1$.
until Termination criterion satisfied.
Output
  Randomly choose an agent l and the value of P_l is the optimal projection matrix P.

B. PDH: Parallel Distributed Hash Algorithm

SDH updates the local estimates $\{P_l\}$ in a cyclic mode. In this section, we propose a PDH algorithm to learn hash functions.

We give a new equivalent bi-convex formulation of (11), as follows:

$$\begin{aligned}\min\ & \sum_{l=1}^m \mathrm{tr}\left(P_l{}^T Q^l P_l\right)\\ \text{s.t.}\ & P_l = Z_{l,t}, \quad Z_{l,t} = P_t, \quad \forall l \in \mathcal{V}, t \in N_l\\ & P_l^T Z_{l,t} = I, \quad Z_{l,t}^{\mathrm{T}} P_t = I, \quad \forall l \in \mathcal{V}, t \in N_l\end{aligned} \tag{22}$$

where $\{Z_{l,t}\}$ are auxiliary variables decoupling the local variable at agent l from those of its neighbors $t \in N_l$. Let $\Lambda_{1,l,t}$ and $\Lambda_{2,l,t}$, respectively, be the Lagrange multipliers corresponding to the constraints $P_l = Z_{l,t}$ and $Z_{l,t} = P_t$, $\forall l \in \mathcal{V}, t \in N_l$. Let $\Gamma_{1,l,t}$ and $\Gamma_{2,l,t}$, respectively, be the Lagrange multipliers corresponding to the constraints $P_l^T Z_{l,t} = I$ and $Z_{l,t}^T P_t = I$, $\forall l \in \mathcal{V}, t \in N_l$. The augmented Lagrangian function of (22) is formulated as

$$\begin{aligned}&L_\rho\left(\{P_l\}, \{Z_{l,t}\}, \left\{\Lambda_{1,l,t}, \Lambda_{2,l,t}\right\}, \left\{\Gamma_{1,l,t}, \Gamma_{2,l,t}\right\}\right)\\ &= \sum_{l=1}^m \mathrm{tr}\left(P_l{}^T Q^l P_l\right)\\ &\quad + \sum_{l=1}^m\sum_{t\in N_l}\left[\mathrm{tr}\left(\Lambda_{1,l,t}^{\mathrm{T}}\left(P_l - Z_{l,t}\right)\right) + \mathrm{tr}\left(\Lambda_{2,l,t}^{\mathrm{T}}\left(Z_{l,t} - P_t\right)\right)\right]\\ &\quad + \frac{\rho}{2}\sum_{l=1}^m\sum_{t\in N_l}\left(\|P_l - Z_{l,t}\|_F^2 + \|Z_{l,t} - P_t\|_F^2\right)\\ &\quad + \sum_{l=1}^m\sum_{t\in N_l}\left[\mathrm{tr}\left(\Gamma_{1,l,t}^{\mathrm{T}}\left(P_l^T Z_{l,t} - I\right)\right) + \mathrm{tr}\left(\Gamma_{2,l,t}^{\mathrm{T}}\left(Z_{l,t}^T P_t - I\right)\right)\right]\\ &\quad + \frac{\rho}{2}\sum_{l=1}^m\sum_{t\in N_l}\left(\|P_l^T Z_{l,t} - I\|_F^2 + \|Z_{l,t}^T P_t - I\|_F^2\right).\end{aligned} \tag{23}$$

Also, let all the Lagrange multipliers be initialized to zeros. Following the ADMM rule, the variables $\{P_l\}, \{Z_{l,t}\}$ and multipliers $\{\Lambda_{1,l,t}, \Lambda_{2,l,t}\}, \{\Gamma_{1,l,t}, \Gamma_{2,l,t}\}$ are updated as follows: for all $l \in \mathcal{V}, t \in N_l$

$$P_l^{k+1} = \left(\tilde{C}_l^k\right)^{-1}\tilde{E}_l^k \tag{24a}$$

$$Z_{l,t}^{k+1} = \left(G_{l,t}^{k}\right)^{-1} H_{l,t}^{k} \tag{24b}$$

$$\Lambda_{1,l,t}^{k+1} = \Lambda_{1,l,t}^{k} + \rho\left(P_{l}^{k+1} - Z_{l,t}^{k+1}\right) \tag{24c}$$

$$\Lambda_{2,l,t}^{k+1} = \Lambda_{2,l,t}^{k} + \rho\left(Z_{l,t}^{k+1} - P_{t}^{k+1}\right) \tag{24d}$$

$$\Gamma_{1,l,t}^{k+1} = \Gamma_{1,l,t}^{k} + \rho\left(\left(P_{l}^{k+1}\right)^{T} Z_{l,t}^{k+1} - I\right) \tag{24e}$$

$$\Gamma_{2,l,t}^{k+1} = \Gamma_{2,l,t}^{k} + \rho\left(\left(Z_{l,t}^{k+1}\right)^{\mathrm{T}} P_{t}^{k+1} - I\right) \tag{24f}$$

where

$$\tilde{C}_{l}^{k} = 2Q^{l} + 2\rho|N_l|I + \rho\sum_{t\in N_l}\left(Z_{l,t}^{k}\left(Z_{l,t}^{k}\right)^{\mathrm{T}} + Z_{t,l}^{k}\left(Z_{t,l}^{k}\right)^{\mathrm{T}}\right) \tag{25}$$

$$\tilde{E}_{l}^{k} = -\sum_{t\in N_l}\left(\Lambda_{1,l,t}^{k} - \Lambda_{2,t,l}^{k}\right) + 2\rho\sum_{t\in N_l}\left(Z_{l,t}^{k} + Z_{t,l}^{k}\right) - \sum_{t\in N_l}\left(Z_{l,t}^{k}\left(\Gamma_{1,l,t}^{k}\right)^{\mathrm{T}} + Z_{t,l}^{k}\Gamma_{2,t,l}^{k}\right) \tag{26}$$

$$G_{l,t}^{k} = 2\rho I + \rho\left(P_{l}^{k+1}\left(P_{l}^{k+1}\right)^{\mathrm{T}} + P_{t}^{k+1}\left(P_{t}^{k+1}\right)^{\mathrm{T}}\right) \tag{27}$$

and

$$H_{l,t}^{k} = \Lambda_{1,l,t}^{k} - \Lambda_{2,l,t}^{k} + 2\rho\left(P_{l}^{k+1} + P_{t}^{k+1}\right) - \left(P_{l}^{k+1}\Gamma_{1,l,t}^{k} + P_{t}^{k+1}\left(\Gamma_{2,l,t}^{k}\right)^{\mathrm{T}}\right). \tag{28}$$

The whole process is described in Algorithm 2. Also, in practice, we simply initialize the variables by random Gaussian matrices. PDH algorithm updates variables in parallel, so it has better time efficiency than SDH. However, the communication, computational, and storage complexities of PDH are higher than those of SDH. So SDH consumes less resources, and is more suitable for low-cost networks.

Remark 1: The theoretical convergence property of the ADMM update rule for the nonconvex optimization is an open question [46]. Like the exiting distributed LH algorithm DisH [50], the algorithms proposed in this paper are also not guaranteed to reach the global optimum. However, in the experiment section, we will empirically show that the proposed algorithms can fast converge to a local minimum, and we also verify the consistency of local projection variables $\{P_l\}$ learned by different agents.

Algorithm 2 PDH Algorithm

Initialization Each agent l obtains q_l anchor points and broadcasts them. Then each agent l computes and broadcasts $\{\varphi_j^l\}_{j=1}^{q}, n_l$. Set the penalty parameter ρ appropriately. Set $\Lambda_{1,l,t}^0 = \Lambda_{2,l,t}^0 = \Lambda_{2,t,l}^0 = 0$, $\forall t \in N_l$. Set $\Gamma_{1,l,t}^0 = \Gamma_{2,l,t}^0 = \Gamma_{2,t,l}^0 = 0$, $\forall t \in N_l$. Initialize the starting points P_l^0, $Z_{l,t}^0$ by random Gaussian matrices, $\forall l \in \mathcal{V}$. Set $k = 0$.

Each agent l computes W^l according to (2).
Each agent l constructs $K(X^l)$ and $K(A)$.
Each agent l computes S^l according to (10).
Each agent l obtains Q^l from S^l.
repeat
  for all $l = 1, 2, \cdots, m$ [in parallel] **do**
    Compute $\tilde{C}_l^k$ and $\tilde{E}_l^k$ using (25) and (26).
    Compute P_l^{k+1} using (24a).
    Transmit P_l^{k+1} to the neighbors of agent l.
  end for
  for all $l = 1, 2, \cdots, m$ [in parallel] **do**
    Compute $G_{l,t}^k$, $H_{l,t}^k$ using (27) and (28), $\forall t \in N_l$.
    Compute $Z_{l,t}^{k+1}$ using (24b), $\forall t \in N_l$.
    Send $Z_{l,t}^{k+1}$ to agent t, $\forall t \in N_l$.
  end for
  for all $l = 1, \cdots, m$ [in parallel] **do**
    Compute $\Lambda_{1,l,t}^{k+1}$ by (24c), $\forall t \in N_l$.
    Compute $\Lambda_{2,l,t}^{k+1}$, $\Lambda_{2,t,l}^{k+1}$ by (24d), $\forall t \in N_l$.
    Compute $\Gamma_{1,l,t}^{k+1}$ by (24e), $\forall t \in N_l$.
    Compute $\Gamma_{2,l,t}^{k+1}$, $\Gamma_{2,t,l}^{k+1}$ by (24f), $\forall t \in N_l$.
  end for
  $k = k + 1$.
until Termination criterion satisfied.
Output
  Randomly choose an agent l, the value of P_l is the optimal projection matrix P.

V. Complexity Analysis

In this section, we give the communication and computational complexity analyses of the proposed two algorithms. Before giving the analysis, we first recall all the related quantities mentioned in this paper: the number of the agents m, the dimensionality of the data d, the number of the anchor points q_l selected by agent l, the number of the total anchor points q, the length of the code c, the size of the data in the lth agent n_l, the degree of the lth agent $|N_l|$. Among these quantities, only n_l is large, and the others are all much smaller than n_l.

Both the communication and computational complexities consist of two parts: 1) the initialization stage and 2) the main procedure. All the operations executed before the main iteration begins are seen as the initialization stage.

A. Communication Complexity

SDH: In the initialization stage, each agent l needs to broadcast its local anchor points. The communication complexity of this step is $\mathcal{O}(q_l d(m-1))$. Then each agent l needs to broadcast $\{\varphi_j^l\}_{j=1}^q$ to obtain the bias $\{v_j\}_{j=1}^q$. The communication complexity of this step is $\mathcal{O}(q(m-1))$. In the main procedure, at each iteration, each agent l needs to transmit P_l to its neighbors. The complexity of this step is $\mathcal{O}(qc|N_l|)$. The total communication complexity of SDH is independent of the data size n_l.

PDH: In the initialization stage, PDH has the same communication complexity as SDH. In the main procedure, at each iteration, each agent l should transmit P_l to its neighbors. The complexity of this step is $\mathcal{O}(qc|N_l|)$. In addition, after computing the auxiliary variables $Z_{l,t}$, each agent l needs to send

TABLE I
COMMUNICATION AND COMPUTATIONAL COMPLEXITIES OF SDH AND PDH

	Step	SDH	PDH
Communication	Broadcast Anchor Points	$\mathcal{O}(q_l d(m-1))$	$\mathcal{O}(q_l d(m-1))$
	Broadcast $\{\varphi_j^l\}_{j=1}^q$	$\mathcal{O}(q(m-1))$	$\mathcal{O}(q(m-1))$
	Transmit P_l	$\mathcal{O}(qc\|N_l\|)$	$\mathcal{O}(qc\|N_l\|)$
	Transmit $Z_{l,t}$	–	$\mathcal{O}(qc\|N_l\|)$
	ToTal	$\mathcal{O}(q_l d(m-1)+q(m-1)+qc\|N_l\|)$	$\mathcal{O}(q_l d(m-1)+q(m-1)+qc\|N_l\|)$
Computation	Step	SDH	PDH
	Compute $W^l, K(X^l)$, and $K(A)$	$\mathcal{O}(dqn_l + dqn_l + dq^2)$	$\mathcal{O}(dqn_l + dqn_l + dq^2)$
	Compute S^l	$\mathcal{O}(2n_l q^2 + q^3)$	$\mathcal{O}(2n_l q^2 + q^3)$
	Compute Q^l	$\mathcal{O}(q^3)$	$\mathcal{O}(q^3)$
	Compute $C_l, E_l(\tilde{C}_l, \tilde{E}_l)$	$\mathcal{O}(q^2 c\|N_l\| + c^2 q\|N_l\|)$	$\mathcal{O}(q^2 c\|N_l\| + c^2 q\|N_l\|)$
	Update P_l	$\mathcal{O}(q^3)$	$\mathcal{O}(q^3)$
	Compute $G_{l,t}, H_{l,t}$	–	$\mathcal{O}(q^2 c\|N_l\| + c^2 q\|N_l\|)$
	Update $Z_{l,t}$	–	$\mathcal{O}(q^3\|N_l\|)$
	Update multipliers	$\mathcal{O}(qc^2\|N_l\|)$	$\mathcal{O}(qc^2\|N_l\|)$
	ToTal	$\mathcal{O}(n_l)$	$\mathcal{O}(n_l)$

$Z_{l,t}$ to its neighbor agent t. The complexity of this step is also $\mathcal{O}(qc|N_l|)$. The overall communication complexity of PDH is higher than SDH, but is still independent of the data size n_l.

As a result, both of the two algorithms have moderate communication complexities, which are independent of n_l.

B. Computation Complexity

SDH: In the initialization stage, for each agent l, the total complexity of initializing W^l, $K(X^l)$, and $K(A)$ is $\mathcal{O}(dqn_l + dqn_l + dq^2)$. The complexity of computing S^l is $\mathcal{O}(2n_l q^2 + q^3)$ and the complexity of computing Q^l is $\mathcal{O}(q^3)$. In the main procedure, for each agent l, at each iteration, the time complexity to compute C_l and E_l is $\mathcal{O}(q^2 c|N_l| + c^2 q|N_l|)$. Then the time complexity to compute P_l using C_l and E_l is $\mathcal{O}(q^3)$. The time complexity of updating all the multipliers is $\mathcal{O}(qc^2|N_l|)$. In summary, in the initialization stage, the time complexity of SDH at each agent is $\mathcal{O}(n_l)$, linear in the data size n_l, and in the main procedure, the time complexity of SDH is independent of n_l. Therefore, SDH is scalable.

PDH: In the initialization stage, PDH has the same time complexity as SDH. In the main procedure, for each agent l, at each iteration, the time complexity to compute $\tilde{C}_l$ and $\tilde{E}_l$ is $\mathcal{O}(q^2 c|N_l| + c^2 q|N_l|)$. The complexity to compute P_l is $\mathcal{O}(q^3)$. The time complexity to compute all the matrix parameters $G_{l,t}$ and $H_{l,t}$ is $\mathcal{O}(q^2 c|N_l| + c^2 q|N_l|)$. The complexity to compute all the auxiliary variables $Z_{l,t}$ is $\mathcal{O}(q^3|N_l|)$. The time complexity of updating all the multipliers is $\mathcal{O}(qc^2|N_l|)$. In summary, in the initialization stage, the time complexity of PDH at each agent is $\mathcal{O}(n_l)$, linear in the data size n_l, and in the main procedure, the time complexity of PDH is independent of n_l. Though the time complexity of PDH is higher than that of SDH, it is still moderate, and PDH is also scalable.

Compared with SDH, PDH consumes more communication and computing resources. In order to store auxiliary variables, PDH also requires more storage resource. However, PDH enables all the agents to update the variables in parallel, while SDH can only update the variables one by one. In summary, PDH is more time efficient, while SDH consumes less resource.

We list the primary computational and communication complexities of SDH and PDH in detail in Table I.

VI. EXPERIMENTS

In this section, we demonstrate the performance of the proposed algorithms by numerical experiments on the problem of image retrieval. In all the experiments, we assume that the data is distributed across a connected network that consists of ten agents and with probability 0.4 a pair of agents are connected to be neighbors. The constructed network has 15 edges, and the maximum degree is 5, as shown in Fig. 1.

We apply our algorithms on four benchmarks: 1) CIFAR-10 [33]; 2) GIST-1M [50]; 3) NUS-WIDE [8]; and 4) TINY-80M [9]. The scale of the first one is relatively small, and the others are all large-scale. The penalty parameter ρ depends on the scale of the training data size, and usually a large value of ρ makes the algorithms have better convergence properties. In the following, we empirically set $\eta' = 5$ and $\rho = 1e10$.

As for the anchor points, generally, the hash function with more anchor points can have better performance and we usually choose more anchor points for larger dataset to better represent the distribution of the whole data. On the other hand, the anchor points will bring additional computation and a moderate number of the anchor points are usually enough to ensure the performance. In real applications, according to the allowable computing resource, one can choose 500–5000 anchor points for training the hash functions. In the following examples, the dataset in Example 1 is relatively small, and we choose 500 anchor points. In the other examples, the datasets are all large-scale, and we choose 1000 anchor points.

All the results below are averaged over 50 independent runs.

A. Results on CIFAR-10

In this example, we test the accuracy performance of the proposed algorithms. We aim to verify that the proposed distributed algorithms are comparable to existing state-of-the-art centralized methods, including PCAH [53], AGH [8], ITQ [3],

TABLE II
MAP WITH DIFFERENT CODE SIZES ON CIFAR-10

Methods	MAP			
	64bits	96its	128bits	256bits
PCAH	0.1237	0.1199	0.1176	0.1131
AGH	0.1419	0.1408	0.1374	0.1326
ITQ	0.1683	0.1758	0.1771	0.1815
ABQ	0.1433	0.1547	0.1517	0.1482
SGH	0.1796	**0.1844**	**0.1890**	**0.1914**
DisH	0.1711	0.1752	0.1769	0.1802
DABQ	0.1429	0.1540	0.1523	0.1477
SDH	**0.1814**	**0.1840**	**0.1861**	**0.1886**
PDH	**0.1809**	**0.1833**	**0.1860**	**0.1889**

ABQ [17], and SGH [31]. We also make a comparison with the existing distributed LH method DisH [50] and DABQ [52] to show the superiority of our methods.

We choose a relatively small benchmark: CIFAR-10 for this experiment. The dataset of CIFAR-10 consists of 60K images in ten classes, and each class has 6K images. Each image is represented by a 512-D GIST descriptor extracted from a 32×32 color image. We randomly choose 1000 data points as the query (test) set, and use the remaining data points for training. Since each data point in CIFAR-10 is assigned a class label, the ground truth neighbors of each query can be easily determined based on label agreement. For DisH, DABQ, and the proposed two distributed algorithms, we evenly divide the training data into ten splits, and each agent collects 1 split. For all compared centralized algorithms, we use a single agent to collect all the training data for learning hash functions. For SDH and PDH, we let each agent randomly select 50 data points as local anchor points, so the number of anchor points is 500 in total. We use the Hamming ranking [8] based evaluation for quantitative performance measurement. For each query, all the data points in the database are ranked (from small to large) according to their Hamming distance to the query.

Table II shows the Hamming ranking perfomance measured by mean average precision (MAP). The code size c varies from 64 to 256 bits. From Table II, we see that SDH and PDH outperform the existing distributed hashing algorithms DisH, DABQ, and most of the state-of-art centralized unsupervised methods except the recently proposed graph hashing method SGH, which demonstrates the effectiveness and efficiency of the proposed methods. Note that the performance of SDH and PDH is very close to that of SGH. It means that the proposed distributed hashing algorithms do not lose much quality compared with the existing centralized graph hashing algorithms.

Then we report the complete precision-recall curves with 256 bits code in Fig. 2. From Fig. 2, we see that SGH and the proposed PDH, SDH outperform other competitors with a large margin, which is consistent with the results shown in Table II.

We report the training time consumption in Table III. From Table III, we see that PDH has the highest time efficiency, and SDH takes the longer time than PDH, but is still efficient. Though SDH is less time efficient, it consumes least resource.

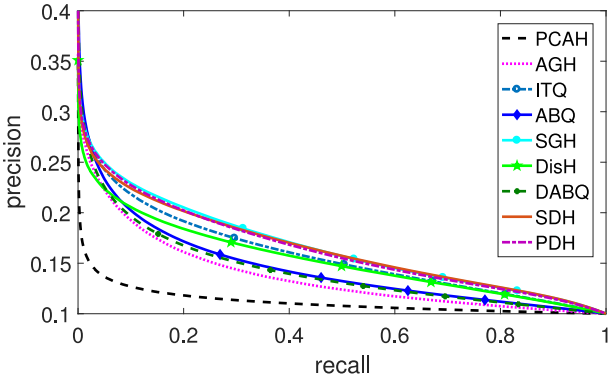


Fig. 2. Precision-recall curves with 256 bits on CIFAR-10.

TABLE III
TRAINING TIME ON CIFAR-10 (IN SECOND)

Methods	Training time (s)			
	64bits	96its	128bits	256bits
PCAH	1.5428	1.7499	2.1176	2.9131
AGH	21.1865	21.6439	22.2375	22.9532
ITQ	2.1643	2.6758	3.1034	3.6472
ABQ	32.6685	38.7364	42.2184	51.3617
SGH	66.4435	88.1378	116.1579	264.2247
DisH	2.1936	2.4576	2.8554	3.0164
DABQ	4.3368	4.6652	5.2588	6.3164
SDH	4.8516	5.5364	6.7721	8.9681
PDH	1.2283	1.8732	2.1489	2.7709

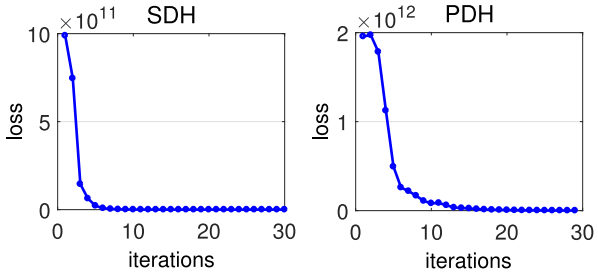


Fig. 3. Loss on CIFAR-10 using SDH and PDH. Both algorithms converge within less than 20 iterations.

Next, we empirically show the convergence and consistency performance of the proposed methods. We respectively choose the Lagrangian functions (15) and (23) as the loss functions for SDH and PDH to demonstrate the convergence of the proposed algorithms. We plot the evolution curves of the loss functions with 64 bits in Fig. 3. Fig. 3 shows that both SDH and PDH converge within less than 20 iterations, which implies that both of the proposed algorithms converge fast in practice.

To show the consistency of SDH and PDH, we define the consistency coefficient

$$\sigma = \sum_{l=1}^{m} \|P_l - \bar{P}\|^2$$

where $\bar{P} = (1/m)\sum_{l=1}^{m} P_l$ is the mean of $P_1, \ldots, P_m$. Note that if $\sigma = 0$, then P_l equals to $\bar{P}$, for all l, which means $P_1, \ldots, P_m$ achieve consensus. For both algorithms, we compute the value of σ with 64 bits at each iteration, and we plot the evolution curves of σ in Fig. 4. Fig. 4 shows that the value

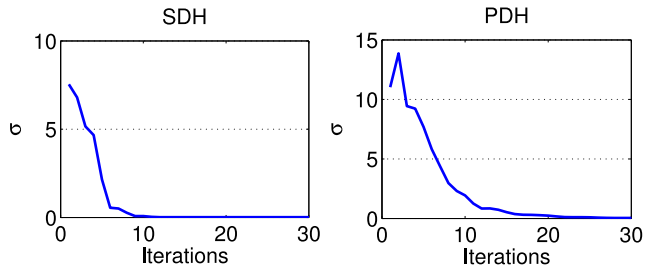


Fig. 4. Evolution of σ using SDH and PDH. The local estimates using SDH achieve consensus within about 10 iterations, and the local estimates using PDH achieve consensus within about 20 iterations.

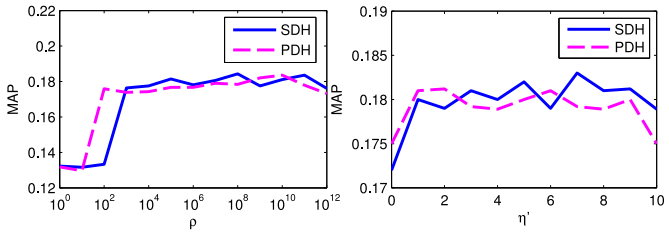


Fig. 5. MAP results with respect to parameters ρ and η' on CIFAR-10.

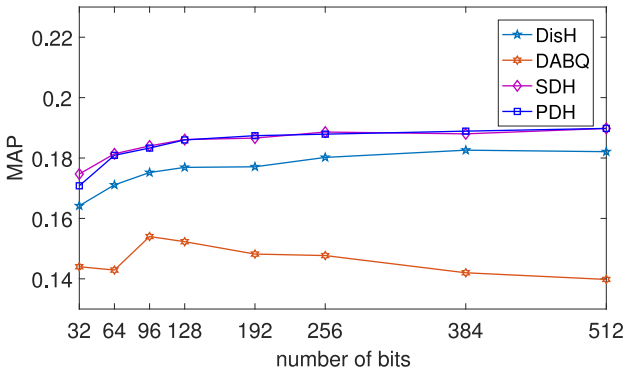


Fig. 6. MAP results with respect to different number of bits on CIFAR-10.

of σ using SDH approximates to 0 within about ten iterations, and the value of σ using PDH approximates to 0 within about 20 iterations, which verifies the consistency of the proposed algorithms. In addition, compared to PDH, SDH has better consistency.

We investigate the sensitivity of the parameters ρ and η'. The MAP results on CIFAR-10 with varying ρ and η' are shown in Fig. 5.

To clearly show the impact the of the code size, we report the performance (MAP) of the distributed hashing methods with respect to the number of the bits in Fig. 6, where the code size c varies from 32-512 bits. We see that, as the number of the bits increases, SDH and PDH show superior advantages compared with DisH and DABQ.

B. Results on GIST-1M

Next, we measure the accuracy of the proposed algorithms and the other methods on a large-scale dataset. The methods for comparison are the same as those in example A. The benchmark for test is GIST-1M, which consists of one million 960-D GIST descriptors extracted from random images. We also randomly choose 1000 data points as query (test) set, and use the remaining data points for training. Since the data points in GIST-1M are not assigned labels, the ground truth is defined as top 2% nearest neighbors in the training set according to the Euclidean distance. Similar to the previous example, for the distributed algorithms, the data points are evenly distributed across ten agents, and for centralized algorithms, all of the data points are collected by a single agent. In this example, each agent randomly selects 100 data points as its local anchor points, and the total number of all anchor points is 1000. Since the dataset of this example is large-scale and computing MAP of the whole dataset is very slow, we only return top 1000 retrieved samples. The MAP result of top 1000 retrieved samples based on Hamming ranking is shown in Table IV. In this example, DABQ performs quite well when the bit length is short and the proposed SDH, PDH methods outperform DABQ when the bit length is long. Later we will further show that as the number of bits increases, the proposed SDH, PDH methods will achieve better performance and show superior advantages compared with DisH and DABQ. The training time consumption is shown in Table V. We see that the proposed algorithms are also time efficient in the large-scale application, especially for PDH, compared with other distributed hashing methods.

Then we plot the precision curves with respect to top 1000 retrieved images with 256 bits code in Fig. 7 to further show the performance of the proposed methods.

We report the performance (Top-1000 MAP) of the distributed hashing methods with respect to the number of the bits in Fig. 8. We see that as the number of the bits increases, SDH and PDH clearly surpass DisH and DABQ. That is in line

TABLE IV
TOP-1000 MAP WITH DIFFERENT CODE SIZES ON GIST-1M

Methods	GIST-1M			
	64bits	96bits	128bits	256bits
PCAH	0.2957	0.2777	0.2631	0.2334
AGH	0.2733	0.2872	0.2815	0.2822
ITQ	0.4918	0.5232	0.5364	0.5527
ABQ	**0.5812**	**0.6457**	0.6282	0.5987
SGH	0.5026	0.5524	0.6173	**0.7144**
DisH	0.4908	0.5117	0.5332	0.5462
DABQ	0.5801	0.6446	**0.6291**	0.5973
SDH	0.4929	0.5472	0.6014	**0.6784**
PDH	0.4934	0.5421	0.5894	**0.7011**

TABLE V
TRAINING TIME ON GIST-1M (IN SECOND)

Methods	GIST-1M			
	64bits	96bits	128bits	256bits
PCAH	2.0608	2.4232	2.9378	3.4268
AGH	113.9523	114.4430	114.8972	115.2657
ITQ	5.4055	6.6807	7.0212	13.8536
ABQ	107.5425	123.4439	158.1759	193.0512
SGH	181.9834	271.7978	410.7303	811.9101
DisH	9.2274	11.8382	14.2214	21.3641
DABQ	15.5861	20.6313	26.3451	33.0608
SDH	25.8487	30.4259	39.5216	47.3839
PDH	6.7432	7.4851	8.6355	11.7164

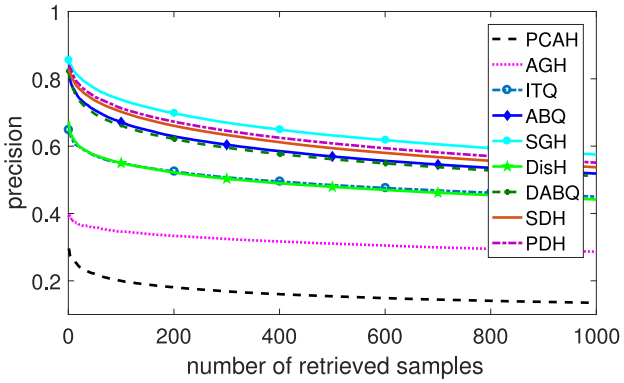


Fig. 7. Precision curves with top 1000 retrieved images with 256 bits on GIST-1M.

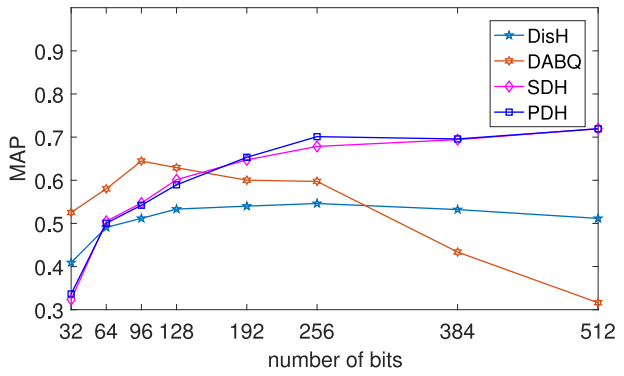


Fig. 8. Top-1000 MAP results with respect to different number of bits on GIST-1M.

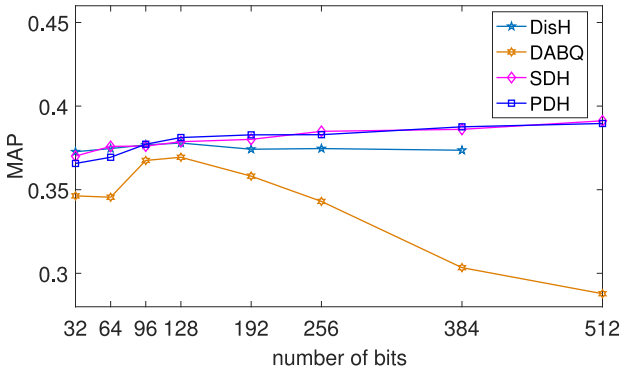


Fig. 9. Top-1000 MAP results with respect to different number of bits on NUS-WIDE.

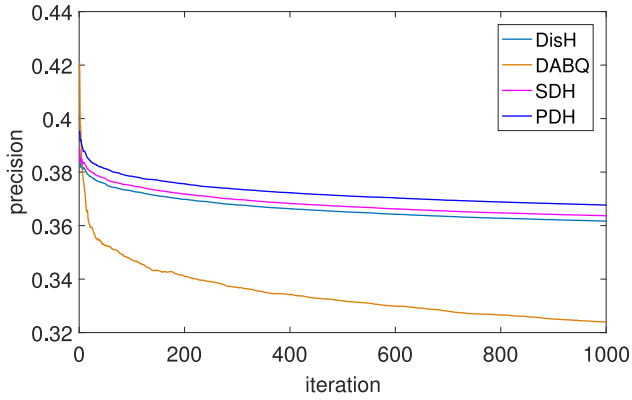


Fig. 10. Precision curves with top 1000 retrieved images with 256 bits on NUS-WIDE.

with the characteristic of graph hashing, which has superior accuracy compared with other methods when the code length is long.

C. Results on NUS-WIDE

We further test the distributed hashing methods DABQ, DisH, SDH, and PDH on the large-scale dataset NUS-WIDE, which contains around 270 000 Web images associated with 81 ground truth concept labels and each image contains multiple semantic labels. All the images are represented by 500-D bag-of-words features. Two images are defined as neighbors if they share at least one label. We collect 21 most frequent labels and for each label, we randomly select 100 images for the query set. So the query set totally has 2100 images. The remaining images are for the training set. We divide the training data points evenly across 10 agents and randomly select 100 data points at each agent as local anchor points, so we totally have 1000 anchor points. We report the Top-1000 MAP result with different code sizes in Fig. 9. The code sizes varies from 32 to 512 bits. For DABQ, SDH, and PDH, they can generate hash codes of any length, while for DisH, the code size cannot be larger than the dimension of the original data, i.e., 500, so we do not give the MAP result of DisH with 512 bits. From the figure, we see that the proposed SDH and PDH still perform well compared with other distributed hashing methods, especially for the long-bit codes.

Then we further illustrate the advantage of the proposed methods by the precision curves with respect to top 1000 retrieved images with 256 bits code, shown in Fig. 10.

D. Results on TINY-80M

At last, we test the distributed hashing methods DisH, DABQ, and the proposed SDH, PDH on a much larger dataset TINY-80M, which consists of 80 million 384-D GIST features. We randomly select 1M data points from the database as the training set and select 1000 queries from the database as the testing set. The ground truth is defined as 5000 nearest neighbors in the database according to the Euclidean distance. We divide the training data points evenly across 10 agents and randomly select 100 data points at each agent as local anchor points, so we totally have 1000 anchor points. Fig. 11 shows the Top-1000 MAP results with respect to different code sizes. The code size varies from 32 to 512 bits. For DisH, the code size cannot be larger than the dimension of the original data, so we also do not give the MAP result with 512 bits using DisH. Similar to the previous examples, the proposed SDH and PDH algorithms perform well and show superior advantages when the bit length is long.

We further use this large-scale example to investigate the impact of the training data size on the performance of the proposed methods. We report the performance (Top-1000 MAP) of the proposed algorithms with respect to different

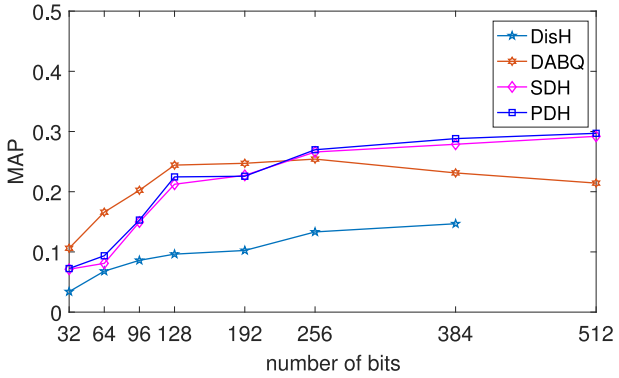


Fig. 11. Top-1000 MAP results with respect to different number of bits on TINY-80M.

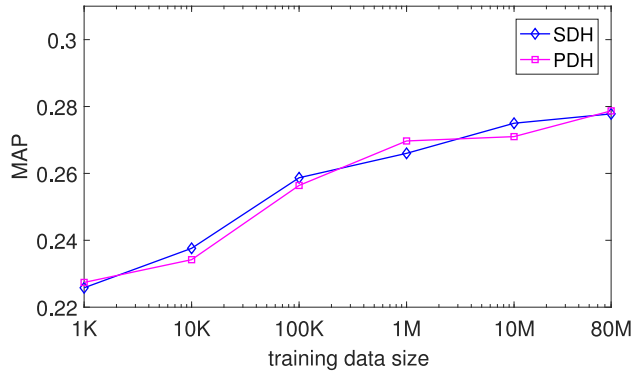


Fig. 12. Top-1000 MAP results with respect to different training data sizes with 256 bits on TINY-80M.

training data sizes with 256 bits in Fig. 12. The size of the training data varies from 1K to 80M. We see that the proposed SDH, PDH algorithms have better performance when more training data points are used. In addition, the proposed algorithms can still have good performance even with very small training data set, so we can use them to solve large-scale retrieval problems if only a small number of training data points are available.

VII. Conclusion

In this paper, we designed a novel distributed hashing model based on graph hashing. We proposed two algorithms to learn efficient hash functions in a distributed manner. The proposed hashing method is suitable for arbitrary network, as long as it is connected. Both of the proposed algorithms were demonstrated to be scalable and have good performance, compared with the existing state-of-art methods.

References

[1] B. Kulis, P. Jain, and K. Grauman, "Fast similarity search for learned metrics," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 12, pp. 2143–2157, Dec. 2009.

[2] L. Liu, M. Yu, and L. Shao, "Multiview alignment hashing for efficient image search," *IEEE Trans. Image Process.*, vol. 24, no. 3, pp. 956–966, Mar. 2015.

[3] 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.

[4] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 21, Vancouver, BC, Canada, 2009, pp. 1753–1760.

[5] Z. Jin *et al.*, "Fast and accurate hashing via iterative nearest neighbors expansion," *IEEE Trans. Cybern.*, vol. 44, no. 11, pp. 2167–2177, Nov. 2014.

[6] X. Liu, J. He, and S.-F. Chang, "Hash bit selection for nearest neighbor search," *IEEE Trans. Image Process.*, vol. 26, no. 11, pp. 5367–5380, Nov. 2017.

[7] R. Zhang, L. Lin, R. Zhang, W. Zuo, and L. Zhang, "Bit-scalable deep hashing with regularized similarity learning for image retrieval and person re-identification," *IEEE Trans. Image Process.*, vol. 24, no. 12, pp. 4766–4779, Dec. 2015.

[8] W. Liu, J. Wang, S. Kumar, and S.-F. Chang, "Hashing with graphs," in *Proc. Int. Conf. Mach. Learn.*, pp. 1–8, 2011.

[9] 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.

[10] 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. 2015.

[11] F. Shen *et al.*, "A fast optimization method for general binary code learning," *IEEE Trans. Image Process.*, vol. 25, no. 12, pp. 5610–5621, Dec. 2016.

[12] X. Zhu, Z. Huang, H. Cheng, J. Cui, and H. T. Shen, "Sparse hashing for fast multimedia search," *ACM Trans. Inf. Syst.*, vol. 31, no. 2, pp. 1–9, 2013.

[13] Y. Cao *et al.*, "Binary hashing for approximate nearest neighbor search on big data: A survey," *IEEE Access*, vol. 6, pp. 2039–2054, 2018, doi: 10.1109/ACCESS.2017.2781360.

[14] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proc. Int. Conf. Very Large Databases*, 1999, pp. 518–529.

[15] L. Liu, M. Yu, and L. Shao, "Unsupervised local feature hashing for image similarity search," *IEEE Trans. Cybern.*, vol. 46, no. 11, pp. 2548–2558, Nov. 2016.

[16] K. He, F. Wen, and J. Sun, "K-means hashing: An affinity-preserving quantization method for learning binary compact codes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Portland, OR, USA, 2013, pp. 2938–2945.

[17] Z. Li, X. Liu, J. Wu, and H. Su, "Adaptive binary quantization for fast nearest neighbor search," in *Proc. ECAI*, 2009, pp. 64–72.

[18] X. Liu, L. Huang, C. Deng, B. Lang, and D. Tao, "Query-adaptive hash code ranking for large-scale multi-view visual search," *IEEE Trans. Image Process.*, vol. 25, no. 10, pp. 4514–4524, Oct. 2016.

[19] C. Deng, X. Tang, J. Yan, W. Liu, and X. Gao, "Discriminative dictionary learning with common label alignment for cross-modal retrieval," *IEEE Trans. Multimedia*, vol. 18, no. 2, pp. 208–218, Feb. 2016.

[20] X. Liu *et al.*, "Multilinear hyperplane hashing," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Las Vegas, NV, USA, 2016, pp. 5119–5127.

[21] J.-P. Heo, Y. Lee, J. He, S.-F. Chang, and S.-E. Yoon, "Spherical hashing," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Providence, RI, USA, 2012, pp. 2957–2964.

[22] 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.

[23] B. Kulis and T. Darrell, "Learning to hash with binary reconstructive embeddings," in *Proc. Adv. Neural Inf. Process. Syst.*, Vancouver, BC, Canada, 2009, pp. 1042–1050.

[24] D. Song, W. Liu, R. Ji, D. A. Meyer, and J. R. Smith, "Top rank supervised binary coding for visual search," in *Proc. IEEE Int. Conf. Comput. Vis.*, Santiago, Chile, Dec. 2015, pp. 1922–1930.

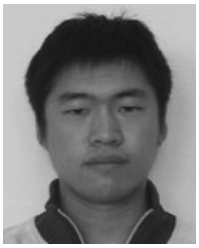
[25] J. Wang, W. Liu, S. Kumar, and S.-F. Chang, "Learning to hash for indexing big data—A survey," *Proc. IEEE*, vol. 104, no. 1, pp. 34–57, Jan. 2016.

[26] G. Lin, C. Shen, Q. Shi, A. van den Hengel, and D. Suter, "Fast supervised hashing with decision trees for high-dimensional data," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Columbus, OH, USA, 2014, pp. 1971–1978.

[27] W. Kong and W.-J. Li, "Isotropic hashing," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1646–1654.

[28] X. Liu, Y. Mu, D. Zhang, B. Lang, and X. Li, "Large-scale unsupervised hashing with shared structure learning," *IEEE Trans. Cybern.*, vol. 45, no. 9, pp. 1811–1822, Sep. 2015.

[29] W. Liu, C. Mu, S. Kumar, and S.-F. Chang, "Discrete graph hashing," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 3419–3427.
[30] H. Liu, R. Ji, Y. Wu, and F. Huang, "Ordinal constrained binary code learning for nearest neighbor search," in *Proc. AAAI Conf. Art. Intell.*, 2017, pp. 2238–2244.
[31] Q. Jiang and W. Li, "Scalable graph hashing with feature transformation," in *Proc. Int. Joint Conf. Artif. Intell.*, 2015, pp. 2248–2254.
[32] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang, "Supervised hashing with kernels," in *Proc. Comput. Vis. Pattern Recognit.*, Providence, RI, USA, 2012, pp. 2074–2081.
[33] F. Shen, C. Shen, W. Liu, and H. T. Shen, "Supervised discrete hashing," in *Proc. Comput. Vis. Pattern Recognit.*, Boston, MA, USA, 2015, pp. 37–45.
[34] J. Gui, T. Liu, Z. Sun, D. Tao, and T. Tan, "Fast supervised discrete hashing," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 2, pp. 490–496, Feb. 2018.
[35] J. Gui, T. Liu, Z. Sun, D. Tao, and T. Tan, "Supervised discrete hashing with relaxation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 3, pp. 608–617, Mar. 2018, doi: 10.1109/TNNLS.2016.2636870.
[36] X. L. Li, B. Veeravalli, and C. C. Ko, "Distributed image processing on a network of workstations," *Int. J. Comput. Appl.*, vol. 25, no. 2, pp. 136–145, 2003.
[37] L. Chen, D. Xu, I. W.-H. Tsang, and X. Li, "Spectral embedded hashing for scalable image retrieval," *IEEE Trans. Cybern.*, vol. 44, no. 7, pp. 1180–1190, Jul. 2014.
[38] E. Yang *et al.*, "Pairwise relationship guided deep hashing for cross-modal retrieva," in *Proc. AAAI Conf. Artif. Intell.*, 2017, pp. 1618–1625.
[39] T. A. Letsche and M. W. Berry, "Large-scale information retrieval with latent semantic indexing," *Inf. Sci.*, vol. 100, nos. 1–4, pp. 105–137, 1997.
[40] S.-Y. Tu and A. H. Sayed, "Diffusion strategies outperform consensus strategies for distributed estimation over adaptive networks," *IEEE Trans. Signal Process.*, vol. 60, no. 12, pp. 6217–6234, Dec. 2012.
[41] A. D. Paola, S. Gaglio, G. L. Re, F. Milazzo, and M. Ortolani, "Adaptive distributed outlier detection for WSNs," *IEEE Trans. Cybern.*, vol. 45, no. 5, pp. 902–913, May 2015.
[42] C. Li, P. Shen, Y. Liu, and Z. Zhang, "Diffusion information theoretic learning for distributed estimation over network," *IEEE Trans. Signal Process.*, vol. 61, no. 16, pp. 4011–4024, Aug. 2013.
[43] J. Hua and C. Li, "Distributed variational Bayesian algorithms over sensor networks," *IEEE Trans. Signal Process.*, vol. 64, no. 3, pp. 783–798, Feb. 2016.
[44] P. A. Forero, A. Cano, and G. B. Giannakis, "Consensus-based distributed support vector machines," *J. Mach. Learn. Res.*, vol. 11, pp. 1663–1707, May 2010.
[45] M. Kim, M. S. Stankovic, K. H. Johansson, and H. J. Kim, "A distributed support vector machine learning over wireless sensor networks," *IEEE Trans. Cybern.*, vol. 45, no. 11, pp. 2599–2611, Nov. 2015.
[46] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, 2011.
[47] E. Camponogara, D. Jia, B. H. Krogh, and S. Talukdar, "Distributed model predictive control," *IEEE Control Syst. Mag.*, vol. 22, no. 1, pp. 44–52, Feb. 2002.
[48] X. Wang, S. Li, and P. Shi, "Distributed finite-time containment control for double-integrator multiagent systems," *IEEE Trans. Cybern.*, vol. 44, no. 9, pp. 1518–1528, Sep. 2014.
[49] R. Olfati-Saber and R. M. Murray, "Distributed cooperative control of multiple vehicle formations using structural potential functions," *IFAC World Congr.*, vol. 15, no. 1, pp. 242–248, 2002.
[50] C. Leng, J. Wu, J. Cheng, X. Zhang, and H. Lu, "Hashing for distributed data," in *Proc. 32nd Int. Conf. Mach. Learn.*, 2015, pp. 1642–1650.
[51] X. Liu, B. Du, C. Deng, M. Liu, and B. Lang, "Structure sensitive hashing with adaptive product quantization," *IEEE Trans. Cybern.*, vol. 46, no. 10, pp. 2252–2264, Oct. 2016.
[52] X. Liu, Z. Li, C. Deng, and D. Tao, "Distributed adaptive binary quantization for fast nearest neighbor search," *IEEE Trans. Image Process.*, vol. 26, no. 11, pp. 5324–5336, Nov. 2017.
[53] Y. Gong and S. Lazebnik, "Iterative quantization: A procrustean approach to learning binary codes," in *Proc. Comput. Vis. Pattern Recognit.*, Colorado Springs, CO, USA, 2011, pp. 817–824.
[54] C. Li and Y. Luo, "Distributed vector quantization over sensor network," *Int. J. Distrib. Sensor Netw.*, vol. 10, Oct. 2014, Art. no. 189619.
[55] P. Shen, C. Li, and Y. Luo, "Distributed vector quantization based on Kullback–Leibler divergence," *Entropy*, vol. 17, no. 12, pp. 7875–7887, 2015.



Shengnan Wang received the B.S. degree in information science and electronic engineering from Zhejiang University, Hangzhou, China, in 2014, where he is currently pursuing the Ph.D. degree with the College of Information Science and Electronic Engineering.

His current research interest includes statistical signal processing and optimization.

Chunguang Li (M'14–SM'14) received the M.S. degree in pattern recognition and intelligent systems and the Ph.D. degree in circuits and systems from the University of Electronic Science and Technology of China, Chengdu, China, in 2002 and 2004, respectively.

He is currently a Professor with the College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou, China. His current research interests include statistical signal processing, machine learning, optimization, and image processing.

Hui-Liang Shen received the B.Eng. and Ph.D. degrees in electronic engineering from Zhejiang University, Hangzhou, China, in 1996 and 2002, respectively.

He was a Research Associate and a Research Fellow with Hong Kong Polytechnic University, Hong Kong, from 2001 to 2005. He is currently a Professor with the College of Information Science and Electronic Engineering, Zhejiang University. His current research interests include multispectral color imaging, image processing, and 3-D computer vision.