Distributed Discrete Hashing by Equivalent Continuous Formulation

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

***Abstract*—Hashing based approximate nearest neighbor search has attracted considerable attention in various fields. Most of the existing hashing methods are centralized, which cannot be used for many large-scale applications with the data stored or collected in a distributed manner. In this article, we consider the distributed hashing problem. The main difficulty of hashing is brought by its inherent binary constraints, which makes the problem generally NP-hard. Most of the existing distributed hashing methods chose to relax the problem by dropping the binary constraints. However, such a manner will bring additional quantization error, which makes the binary codes less effective. In this paper, we propose a novel distributed discrete hashing method, which learns effective hash codes without using any relaxations. Specifically, we give a method to transform the discrete hashing problem into an equivalent distributed continuous optimization problem. After transformation, we devise a distributed discrete hashing (dDH) algorithm based on the idea of DC programming to solve the problem. To obtain more efficient hash codes, we further add bits balance and uncorrelation constraints to the hashing problem, and we also propose a distributed constrained discrete hashing algorithm (dCDH) to solve this problem. Extensive experiments are provided to show the superiority of the proposed methods.**

***Index Terms*—Learning to hash, distributed hashing, discrete hashing, large-scale image retrieval, DC programming.**

I. Introduction

RECENTLY, HASHING based approximate nearest neighbor search has a wide range of applications in image retrieval [1], [2], computer vision [3], [4], and machine learning [5], [6], etc. By encoding high-dimensional data points into compact binary codes, hashing can accomplish nearest neighbor search with a constant time complexity. An effective hashing method should be similarity preserving [11], namely, it should map similar data points to adjacent binary hash codes.

Manuscript received May 2, 2019; revised November 15, 2019 and January 12, 2020; accepted January 27, 2020. Date of publication February 20, 2020; date of current version March 10, 2020. This work was supported in part by the National Natural Science Foundation of China under Grants 61571392 and 61631003, in part by the National Key R&D Program of China under Grant 2018YFE0125400, and in part by the National Program for Special Support of Eminent Professionals. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Paolo Di Lorenzo. *(Corresponding author: Chunguang Li.)*

Shengnan Wang and Hui-Liang Shen are with the College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou 310027, China (e-mail: wsn511799@163.com; shenhl@zju.edu.cn).

Chunguang Li is with the College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou 310027, China, and also with the Ningbo Research Institute, Zhejiang University, Ningbo 315100, China (e-mail: cgli@zju.edu.cn).

Digital Object Identifier 10.1109/TSIPN.2020.2975356

Generally, the hashing methods can be mainly divided into data-independent hashing [6], [16], [17] and data-dependent hashing [7], [11]–[13]. Locality sensitive hashing (LSH) [16], [23] should be one of the most popular data-independent hashing methods. LSH generates hash functions by random projections and it can map similar samples to similar codes with a high probability. However, it usually requires long bits to achieve good precision. Recently, more works are related to the data-dependent hashing, which learns binary codes and hash functions based on data. Due to this, the data-dependent hashing methods are also called the learning to hash (LH) methods [24]. The LH methods can effectively and efficiently map massive data points to very short compact binary codes. Representative LH methods include the unsupervised hashing methods: spectral hashing [11], ITQ [8], adaptive binary quantization hashing [9], structure sensitive hashing [15], discrete graph hashing [18]; and supervised hashing methods: minimal loss hashing (MLH) [19], supervised discrete hashing (SDH) [13], supervised hashing with kernels (KSH) [12], latent factor hashing (LFH) [32], column sampling based discrete supervised hashing [29], and so on. The main difference between unsupervised and supervised hashing is whether the label information of data is available for learning binary codes and hash functions.

More recently, some deep hashing methods [20]–[22] were proposed, which use deep networks to learn binary hash codes and achieved promising performance. However, deep hashing is much more time-consuming in both training and testing stages, compared with the non-deep hashing, which is inefficient in practice and may restrict its application in fast similarity search [9], [14], [30]. One of the future directions of deep hashing is to design a proper hashing technique to accelerate deep neural network training and save memory space. In this work, we mainly consider the non-deep hashing for fast approximate nearest neighbor search.

Most of the existing LH methods are centralized, so they can only be implemented in a single machine. However, in many real problems, such as the applications of search engine [33], mobile surveillance [34], and sensor networks [35], the data are often distributed across different locations. Besides, in practice, we usually need to deal with some big-data problems, such as large-scale image processing [36], [37] and large-scale information retrieval [38], which are far beyond the capacity of a single machine (computer). In this situation, we usually partition the data and store them in multiple machines. As a consequence, it is of great importance to develop distributed hashing algorithms to deal with these problems. Some pioneering work has been done, such as the distributed ITQ [39], Hashing for distributed data (DisH) [26], and distributed graph hashing [28]. The inherent binary constraints make the hashing problem NP-hard, and for tractability, the above distributed hashing methods chose to relax the problem by eliminating the binary constraints. Such a manner greatly simplifies the problem. However, it also causes large quantization error and makes the learned binary codes less effective [13]. To obtain high-quality binary codes, some discrete hashing methods [18], [29], [30] were proposed, which solve the hashing problem without using any relaxations. However, these methods are all centralized, which cannot be used for distributed hashing. Directly extending the existing centralized discrete hashing methods into the distributed version is difficult, for the convergence and consensus cannot be guaranteed. The main reason is that in the area of distributed optimization, all the algorithms and theory are based on the condition of continuity. Distributed discrete optimization is an open question so far.

In this paper, we consider the distributed hashing problem. We cast a distributed hashing model, in which the objective function is decomposed into several local objective functions. To guarantee the effectiveness of the binary codes, we aim to solve the distributed hashing problem without using any relaxations. The main contributions of this paper are as follows:

1) We propose a distributed discrete hashing method to solve the distributed hashing problem. This is the first unsupervised distributed discrete hashing method, which does not use any relaxations so that no quantization errors will be introduced. To address the difficulty brought by the binary constraints and solve the problem in a distributed manner, we propose a method to transform the discrete hashing problem into an equivalent distributed continuous optimization problem.
2) The bits balance and uncorrelation constraints can make the hash codes efficient, however, they also make the problem difficult to be solved in a distributed manner. The existing distributed hashing methods use the relaxation manner to deal with this problem. In this paper, we propose a method to address the difficulty brought by the two constraints and reformulate the constrained hashing problem into the form of a tractable distributed optimization problem, without using any relaxations.
3) We propose distributed algorithms based on the idea of DC programming to solve the transformed distributed continuous optimization problems.

It is worth mentioning that in this paper we mainly consider the distributed unsupervised discrete hashing. However, the proposed distributed discrete hashing can be easily extended to the supervised scenario by replacing the unsupervised hashing objective with a supervised hashing objective. We give an example of distributed supervised discrete hashing in the Appendix.

The rest of this paper is organized as follows. In Section II, we introduce some preliminary knowledges. In Section III, we cast a distributed hashing model and propose a distributed discrete hashing algorithm. In Section IV, we further give the method to address the distributed hashing problem with bits balance and uncorrelation constraints. In Section V, we give the method for obtaining hash functions in a distributed manner. The communication and computational complexity analyses are given in Section VI and experiments on large-scale benchmark datasets are provided in Section VII. Finally, we draw the conclusion in Section VIII. An example of distributed supervised discrete hashing is presented in the Appendix A.

II. Preliminaries

A. Notations

In this paper, we use a lowercase letter to denote a column vector and a capital letter to denote a matrix. For a vector x (or a matrix X), we use x^T (or X^T) to denote its transpose. For a vector x, we use $||x||$ to denote the l_2 norm of x. For a matrix X, we use $||X||$ to denote the Frobenius norm of X. For a closed set Ω and a vector x, we use $\mathcal{P}_\Omega[x]$ to denote the projection of x into Ω. We use ∇f to denote the gradient (or subgradient) of a convex function $f(\cdot)$, and we use $\text{sign}(\cdot)$ to denote the element-wise sign function. We use $tr(X)$ to denote the trace of a matrix X. For each agent l, we use $\mathcal{N}_l$ to denote the set of is neighbors, and we use $|\mathcal{N}_l|$ to denote the degree of agent l.

B. Distributed 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. Graph hashing maps the data points into binary codes by solving the following hashing problem

$$\begin{aligned} \min_B \quad & tr(B(S-W)B^T) \\ \text{s.t.} \quad & B \in \{-1,1\}^{r\times n}, \end{aligned} \tag{1}$$

where $W = [W_{i,j}]_{n\times n}$ is called the graph matrix with $W_{i,j} = \exp(-||x_i - x_j||^2/\epsilon)$, and S is a diagonal $n \times n$ matrix with $S_{i,i} = \sum_j W_{i,j}$. The matrix B denotes the binary codes of the dataset X. The parameter $\epsilon > 0$ defines the distance in $\mathbb{R}^d$ which corresponds to similar items. Note that the time complexity to compute W is $\mathcal{O}(dn^2)$, which is unacceptable in large-scale applications. In practice, we usually use the method in [7] to construct the graph matrix by $W = UU^T$, where $U \in \mathbb{R}^{n\times p} (p \ll n)$ is a truncated similarity matrix which is highly sparse. The time complexity to construct W by U is only $\mathcal{O}(pdn)$.

In [28], based on graph hashing, we proposed a distributed hashing model for learning hash functions in a distributed manner. In the distributed scenario, the data are stored across m agents (machines) over a connected network. 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. Then the whole data X is a concatenation of local datasets, i.e., $X = [X^1, X^2, \ldots, X^m]$. As stated in [28], in the distributed scenario, it is difficult to directly construct the graph matrix $W = [W_{i,j}]_{n\times n}$ as shown in (1) for modeling the distributed hashing problem, due to unacceptable communication and computation cost. For tractability, in [28], we constructed an anchor point based graph matrix $\tilde{W} \in \mathbb{R}^{n\times q}$ with

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

where $a_1, \ldots, a_q$ are common anchor points. In [28], we have given the method to obtain the anchor points satisfying the distributed setting. The graph matrix $\tilde{W}$ stores the similarity between the data points and anchor points. It can also 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. The distributed graph hashing in [28] is formulated as follows

$$\begin{aligned} \max_{B,Z} \quad & \sum_{l=1}^{m} tr(B^l \tilde{W}^l Z^T), \\ \text{s.t.} \quad & B^l \in \{-1,1\}^{r \times n_l}, Z \in \{-1,1\}^{r \times q}, \end{aligned} \tag{3}$$

where B^l denotes the hash code matrix of the data X^l and Z denotes the hash code matrix of the anchor points. In [28], we gave two distributed algorithms to solve the distributed graph hashing problem.

However, generally the hashing problem is a nonconvex mixed-integer optimization problem, due to the discrete constraints, which is NP-hard. The algorithms proposed in [28] solve the distributed graph hashing problem by relaxation, which will make the learned hash codes of low quality. In fact, most of the existing distributed hashing methods adopted the relaxation method to address the hashing problem. In addition, the distributed graph hashing model (3) does not fully utilize the available similarity information, so the performance of the distributed graph hashing can be further improved.

C. DC Programming and DC Algorithm

Here, we give a brief introduction of DC (difference of convex functions) programming and DC algorithm (DCA) [44], [45], which may be used in the following paper. DC programming approach is widely used for dealing with smooth or nonsmooth nonconvex continuous optimization problem. The main idea of DC programming is to represent the nonconvex objective function $f(x)$ by the difference of convex functions, namely,

$$\min f(x) = g(x) - h(x), x \in \mathcal{C}, \tag{4}$$

where both $g(x)$ and $h(x)$ are convex functions, and $\mathcal{C} \subseteq \mathbb{R}^d$ is the domain.

DCA is designed for solving the DC programming (4). The existing experiment results showed that DCA quite often gives a global optimal solution, and DCA is proved to be more robust and more efficient than other related standard methods, especially when the problem is large-scale [43]–[45]. DCA solves the DC programming (4) by using the following update rule

$$\begin{cases} y := \nabla h(x), \\ x := \arg\min\{g(x) - \langle x, y\rangle; x \in \mathcal{C}\}, \end{cases} \tag{5}$$

where y is called the dual variable, and $\nabla h(x)$ denotes the subgradient of $h(x)$. If $h(x)$ is differentiable, $\nabla h(x)$ is just the gradient of $h(x)$.

III. Distributed Discrete Hashing

A. Distributed Discrete Hashing Model

In this section, we aim to fully utilize the similarity information among the distributed data and give an effective distributed discrete hashing method to generate high-quality hashing codes. The scenario is the same as that shown in Section II-B. The data are distributed across m agents over a connected network, and each agent l owns its local dataset X^l. First, we will also generate the anchor points $a_1, \ldots, a_q$, which are known by all the agents. Utilizing the anchor points and the local data $X^l = [x_1^l, x_2^l, \ldots, x_{n_l}^l]$, each agent l can construct a local graph matrix $W^l \in \mathbb{R}^{(n_l+q)\times(n_l+q)}$. Here, we will use the method in [7] mentioned above to finish this task, namely $W^l = U^l (U^l)^T$ with $U^l \in \mathbb{R}^{(n_l+q)\times p} (p \ll n_l)$.

Now we give the distributed discrete hashing model, as follows

$$\begin{aligned} \min_{B,Z} \quad & \sum_{l=1}^{m} tr([B^l, Z](S^l - W^l)[B^l, Z]^T), \\ \text{s.t.} \quad & B^l \in \{-1,1\}^{r \times n_l}, Z \in \{-1,1\}^{r \times q}, \end{aligned} \tag{6}$$

where B^l stores the hash codes of the local data X^l, and Z stores the hash codes of the anchor points. For each $l \in \{1, \ldots, m\}$, the matrix S^l is a diagonal $(n_l + q) \times (n_l + q)$ matrix with $S_{i,i}^l = \sum_j W_{i,j}^l$. In (6), the local objective function $L_l(B^l, Z) = tr([B^l, Z](S^l - W^l)[B^l, Z]^T)$ is known by agent l only. Compared with the distributed graph hashing model (3), in (6), more similarity information is utilized, not only the similarity information between the data points and the anchor points, but also the similarity information among the local data points from the same agent, as well as the similarity information among the anchor points.

More importantly, as stated before, the existing distributed hashing algorithms solve the hashing problem by using relaxations, which will cause large quantization error and make the hash code less effective. To obtain high-quality hash codes, in the following, we will propose a distributed hashing method to solve the hashing problem (6) without using any relaxations.

B. Equivalent Continuous Optimization Transformation

Problem (6) is a distributed discrete optimization problem. Directly optimizing such a problem is very difficult, especially in a distributed manner, due to the lack of distributed algorithms for discrete optimization. In the literature, most of the existing distributed optimization algorithms are related to continuous optimization.

In [47], we gave a method to transform the centralized hashing problem into an equivalent continuous optimization problem, and we showed that the hashing problem can be well solved after transformation. In the following, we aim to transform the distributed hashing problem (6) into an equivalent distributed continuous optimization problem, so that we can devise distributed continuous optimization algorithms to solve the problem. Before giving the transformation method, we need to introduce the following lemma.

Lemma 1 ([42]): Let f be a Lipschitz continuous function on $\mathcal{X}$ with constant $\mathcal{L}$. Let φ be a nonnegative function defined on $\mathcal{X}$ and $\mathcal{S} := \{x \in \mathcal{X} : \varphi(x) = 0\}$. If $d(x, \mathcal{S}) \le \varphi(x)$ for all $x \in \mathcal{X}$, then the two problems

$$\inf\{f(x) : x \in \mathcal{S}\} \quad \text{and} \quad \inf\{f(x) + \gamma\varphi(x) : x \in \mathcal{X}\}$$

are equivalent with $\gamma > \mathcal{L}$. In other words, the two problems have the same optimal solutions and the same optimal value. Here,

$$d(x, \mathcal{S}) := \inf_{z \in \mathcal{S}} ||x - z||,$$

which is the distance from a point x to the set $\mathcal{S}$.

Lemma 1 implies that by constructing suitable penalty function $\varphi(\cdot)$, the domain of the optimization problem can be enlarged while the optimal solution of the problem is unchanged. The key point is to find or construct such a penalty function $\varphi(\cdot)$. In [47], we have constructed an effective penalty function for the hashing problem. We showed that a general hashing problem

$$\begin{aligned} \min_{B} \quad & L(B) \\ \text{s.t.} \quad & B \in \{-1, 1\}^{r \times n} \end{aligned} \tag{7}$$

is equivalent to the following continuous optimization problem

$$\begin{aligned} \min_{B} \quad & L(B) + \gamma\varphi(B) \\ \text{s.t.} \quad & B \in [-1, 1]^{r \times n}, \end{aligned} \tag{8}$$

with $\varphi(B) = rn - tr(BB^T)$ and $\gamma > \mathcal{L}$, where $\mathcal{L}$ is the Lipschitz constant of $L(B)$ on $[-1, 1]^{r \times n}$. The penalty term $\gamma\varphi(B)$ forces the optimal solution of problem (8) to be in $\{-1, 1\}^{r \times n}$. The specific process and detailed proof can be seen in [47].

Therefore, according to [47], we can show that problem (6) is equivalent to the following continuous optimization problem

$$\begin{aligned} \min_{B,Z} \quad & \sum_{l=1}^{m} tr([B^l, Z](S^l - W^l)[B^l, Z]^T) + \gamma\varphi([B, Z]), \\ \text{s.t.} \quad & B^l \in [-1, 1]^{r \times n_l}, Z \in [-1, 1]^{r \times q}, \end{aligned} \tag{9}$$

where $\varphi([B, Z]) = r(n + q) - tr([B, Z][B, Z]^T)$ and $\gamma > \mathcal{L}'$. Here, $\mathcal{L}'$ is the Lipschitz constant of the objective function in (6) with respect to the whole variable $[B, Z]$ on $B \in [-1, 1]^{r \times n_l}$ and $Z \in [-1, 1]^{r \times q}$. However, though the above problem is a continuous optimization problem, it is not the standard form of distributed optimization. The objective function is not represented as the sum of local objective functions. So it is difficult to apply the distributed optimization algorithms to such a problem. In addition, it is not easy to know the Lipschitz constant of the objective function with respect to the whole variable $[B, Z]$ in the distributed setting. Hence, more reasonable transformation is needed.

In the following, we transform the problem (6) into an equivalent distributed continuous optimization problem step by step, satisfying the distributed setting. Note that each local variable B^l only exists in agent l, so each agent l can independently construct a local penalty function $\varphi_l(B^l) = rn_l - tr(B^l(B^l)^T)$. Then according to [47], problem (6) can be first transformed to the following problem

$$\begin{aligned} \min_{B,Z} \quad & \sum_{l=1}^{m} \left[tr([B^l, Z](S^l - W^l)[B^l, Z]^T) + \gamma_l\varphi_l(B^l)\right], \\ \text{s.t.} \quad & B^l \in [-1, 1]^{r \times n_l}, Z \in \{-1, 1\}^{r \times q}, \end{aligned} \tag{10}$$

with $\gamma_l > \mathcal{L}^l$, where $\mathcal{L}^l$ is the Lipschitz constant of the local objective function $L_l(B^l, Z)$ with respect to the local variable B^l, which can be easily obtained by agent l. Next, we address the discrete global variable Z. Similarly, by defining $\varphi(Z) = rq - tr(ZZ^T)$, the problem (10) can be further transformed to

$$\begin{aligned} \min_{B,Z} \quad & \sum_{l=1}^{m}[tr([B^l, Z](S^l - W^l)[B^l, Z]^T) \\ & + \gamma_l\varphi_l(B^l)] + \gamma_z\varphi(Z), \\ \text{s.t.} \quad & B^l \in [-1, 1]^{r \times n_l}, Z \in [-1, 1]^{r \times q}, \end{aligned} \tag{11}$$

with $\gamma_z > \mathcal{L}_z$, where $\mathcal{L}_z$ denotes the Lipschitz constant of the whole objective function of (10) with respect to Z. So the only thing is to find a constant larger than $\mathcal{L}_z$. Note that each agent l can easily obtain the Lipschitz constant of the local objective function $L_l(B^l, Z)$ with respect to Z, denoted by $\mathcal{L}_z^l$. Then a simple way is that each agent l broadcasts $\mathcal{L}_z^l$, and all the agents set $\gamma_z = m \max\{\mathcal{L}_z^l\}$.

We further represent the problem (11) as follows

$$\begin{aligned} \min_{B,Z} \quad & \sum_{l=1}^{m}[tr([B^l, Z](S^l - W^l)[B^l, Z]^T) + \gamma_l\varphi_l(B^l) \\ & + \frac{1}{m}\gamma_z\varphi(Z)], \\ \text{s.t.} \quad & B^l \in [-1, 1]^{r \times n_l}, Z \in [-1, 1]^{r \times q}, \end{aligned} \tag{12}$$

and we use $f_l(B^l, Z) = tr([B^l, Z](S^l - W^l)[B^l, Z]^T) + \gamma_l\varphi_l(B^l) + \frac{1}{m}\gamma_z\varphi(Z)$ to denote the new local objective function of agent l. Therefore, we have successfully transformed the distributed hashing problem into an equivalent distributed continuous optimization problem.

Remark 1: In Lemma 1, the condition $\gamma > L$ is only a sufficient condition but not necessary. In the above, we have given the method to obtain the parameter γ that theoretically guarantees the equivalence of the transformed problem and the original problem. In practice, usually, a much smaller γ is enough to guarantee the equivalence, especially for the large-scale problem in which the Lipschitz constant $\mathcal{L}$ is usually very large. In [47], we have shown that in the large-scale hashing problem, a moderate γ is enough to guarantee that the transformed continuous hashing problem has the same optimal solution as the original discrete hashing problem. In the experiment section of this paper, we will also show similar results.

C. Distributed Learning Algorithm

The transformed problem is still a nonconvex optimization problem, since the penalty functions $\{\varphi_l(B^l)\}$ and $\varphi_z(Z)$ are all concave functions. However, the continuous property makes it possible to solve such a problem in a distributed manner. We aim to adopt the idea of DC programming and DCA to solve this problem. First of all, we should rewrite the problem as the form of DC programming. Though the problem (12) can be rewritten as many DC programmings, since each agent l only knows its local objective function in (12), we should ensure that the whole process satisfies the distributed setting. To finish this task, we first let each agent decompose its local objective function $f_l(B^l, Z)$ as the difference of convex functions, as follows,

$$f_l(B^l, Z) = g_l(B^l, Z) - h_l(B^l, Z),$$

with

$$g_l(B^l, Z) = tr([B^l, Z](S^l - W^l)[B^l, Z]^T)$$

and

$$h_l(B^l, Z) = -\gamma_l \varphi_l(B^l) - \frac{1}{m}\gamma_z \varphi(Z).$$

It can be verified that both $g_l(B^l, Z)$ and $h_l(B^l, Z)$ are convex. Then the whole problem (12) can be represented as the following DC programming

$$\begin{aligned} \min_{B,Z} \quad & G(B, Z) - H(B, Z), \\ \text{s.t.} \quad & B \in [-1, 1]^{r\times n}, Z \in [-1, 1]^{r\times q}, \end{aligned} \tag{13}$$

with $G(B, Z) = \sum_{l=1}^m g_l(B^l, Z)$ and $H(B, Z) = \sum_{l=1}^m h_l(B^l, Z)$. For simplicity, we use E to denote the whole variable of the problem, namely $E = [B, Z]$. Then problem (13) can also be represented as

$$\begin{aligned} \min_{B,Z} \quad & G(E) - H(E), \\ \text{s.t.} \quad & E \in [-1, 1]^{r\times(n+q)}. \end{aligned} \tag{14}$$

DCA solves such a DC programming (14) by repeating the following two steps

$$A = \nabla H(E), \tag{15}$$

$$E = \arg\min_{E\in[-1,1]^{r\times(n+q)}} \{G(E) - tr(E^T A)\}, \tag{16}$$

where $\nabla H(E) = [\frac{\partial H}{\partial B}, \frac{\partial H}{\partial Z}]$, and $\frac{\partial H}{\partial B} = [\frac{\partial H}{\partial B^1}, \ldots, \frac{\partial H}{\partial B^m}]$. Let

$$A^l = \frac{\partial H}{\partial B^l} = \frac{\partial h_l}{\partial B^l} = 2\gamma_l B^l$$

for all $l \in \{1, \ldots, m\}$, and let

$$A_z = \frac{1}{m}\frac{\partial H}{\partial Z} = \frac{2\gamma_z}{m} Z.$$

One can see that A^l can be independently computed by agent l, and A_z can be computed by all the agents. Note that

$$\begin{aligned} tr(E^T A) &= \sum_{l=1}^m tr((B^l)^T A^l) + m tr(Z^T A_z) \\ &= \sum_{l=1}^m [tr((B^l)^T A^l) + tr(Z^T A_z)]. \end{aligned}$$

Then problem (16) can be represented as the following distributed convex optimization problem

$$\begin{aligned} \min_{B,Z} \quad & \sum_{l=1}^m [g_l(B^l, Z) - tr((B^l)^T A^l) - tr(Z^T A_z)] \\ \text{s.t.} \quad & B^l \in [-1, 1]^{r\times n_l}, Z \in [-1, 1]^{r\times q}, \end{aligned} \tag{17}$$

Problem (17) is a special distributed optimization problem, in which $\{B^l\}$ are local variables and Z is the global variable. Since the problem is a convex optimization problem, we can use the efficient projected gradient descent method to obtain the optimal solution. Let $J(B, Z)$ denote the whole objective function of problem (17), and let $J_l(B^l, Z) = g_l(B^l, Z) - tr((B^l)^T A^l) - tr(Z^T A_z)$, which is the local objective function of agent l. Note that the gradient of the whole objective function with respect to the local variable B^l can be computed as

$$\frac{\partial J}{\partial B^l} = \frac{\partial J_l}{\partial B^l},$$

which can be independently obtained by agent l. While the gradient of the whole objective function with respect to the global variable Z is

$$\frac{\partial J}{\partial Z} = \sum_{l=1}^m \frac{\partial J_l}{\partial Z},$$

which is difficult to obtain since each agent l only knows J_l. So we aim to use the distributed projected gradient descent [48], [50] method to deal with the global variable Z. Let Z^l denote the local estimation of the global variable Z at agent l. To solve the problem, at each iteration, each agent l first computes the gradient of $J_l(B^l, Z^l)$ with respect to the variable $[B^l, Z^l]$, which is

$$\nabla J_l(B^l, Z^l) = 2[B^l, Z^l](S^l - W^l) - [A^l, A_z^l],$$

where $A_z^l = \frac{2\gamma_z}{m} Z^l$. The update of the local variable B^l can be directly finished by agent l using the projected gradient descent step without any communication with other agents. While the update of Z^l includes two steps. First, each agent l obtains an intermediate estimate R^l using the local projected gradient descent. Then each agent l shares its intermediate estimate R^l with its neighbors and updates Z^l as the weighted average of all the obtained intermediate estimates (including R^l). The whole process is shown in Algorithm 1.

Here, $\mathcal{D}_l := [-1, 1]^{r\times(n_l+q)}$, which are the projection sets, $w_{lj} \geq 0$ is the weight agent l assigns to the information received from agent j, and $w_{lj} > 0$ if and only if $j = l$ or $j \in \mathcal{N}_l$. The weight parameters $\{w_{lj}\}$ should satisfy that $\sum_{j=1}^m w_{lj} = 1, \forall l \in \{1, \ldots, m\}$, and $\sum_{l=1}^m w_{lj} = 1, \forall j \in \{1, \ldots, m\}$. The method for generating such weight parameters satisfying all the above requirements can be seen in [51]. According to [25], [48], [50], repeating the projected gradient descent step and communication step, the local estimates $\{Z^l\}$ will achieve consensus, and the variables will converge to the optimal solution of problem (17). The convergence proof of the projected gradient descent method and distributed projected gradient descent method can

Algorithm 1: Distributed Projected Gradient Descent (dPGD) Algorithm for Subproblem (17).

```
Input: $B^l \in \mathbb{R}^{r\times n_l}$, $Z^l \in \mathbb{R}^{r\times q}$, $A^l \in \mathbb{R}^{r\times n_l}$, and
    $A_z^l \in \mathbb{R}^{r\times q}$, for all $l \in \{1,\ldots,m\}$.
Repeat
    Computing gradient step: each agent $l$ compute the
     gradient
        $\nabla J_l(B^l, Z^l) = 2[B^l, Z^l](S^l - W^l) - [A^l, A_z^l];$
    Projected gradient descent step: Each agent $l$ computes
        $[B^l, R^l] = \mathcal{P}_{\mathcal{D}_l}([B^l, Z^l] - \alpha \nabla J_l(B^l, Z^l));$
    Communication step: Each agent $l$ shares $R^l$ with all the
     neighbors $j \in \mathcal{N}_l$ and updates $Z^l$ by
        $Z^l = \sum_{j=1}^{m} w_{lj} R^j;$
Until convergence condition holds.
Output: $B^l \in \mathbb{R}^{r\times n_l}$, $Z^l \in \mathbb{R}^{r\times q}$, for all $l \in \{1,\ldots,m\}$.
```

Algorithm 2: dDH.

```
Input the parameters $\{\gamma_l\}, \gamma_z$ and anchor points $\{a_i\}_{i=1}^q$.
 Each agent $l$ constructs the $W^l$ and $S^l$ using the local data
 points and anchor points. Initialize the variables $B^l, Z^l$ by
 the sign of random Gaussian matrices.
Loop until convergence or reach $T$ times
    Each agent $l$ computes
        $A^l = 2\gamma_l B^l, A_z^l = \frac{2\gamma_z}{m} Z^l;$
    Each agent $l$ updates $[B^l, Z^l]$ by
        $\{B^l, Z^l\} = \text{dPGD}(\{B^l\}, \{Z^l\}, \{A^l\}, \{A_z^l\}).$
Output The binary codes $\{B^l\}$.
```

be seen in [25], [48]. To avoid repetition, we do not give detailed proof here.

We summarize the whole process of the proposed distributed discrete hashing (dDH) algorithm for solving the hashing problem (12) in Algorithm 2.

IV. Distributed Discrete Hashing With Bits Balance and Uncorrelation Constraints

A. Problem Formulation

The bits balance and uncorrelation constraints were widely used in the existing centralized hashing methods to improve the efficiency of hash codes. However, the two constraints will make the hashing problem more complex and difficult to be solved in a distributed manner. So the existing distributed hashing methods either did not consider the two constraints or relaxed the two constraints during the optimization stage, which makes the hash codes less efficient as they are expected.

In the following, we add the bits balance and uncorrelation constraints in the proposed distributed hashing model (6), and we still aim to solve the problem without using any relaxations to ensure the effectiveness and efficiency of hash codes. With the bits balance and uncorrelation constraints, the problem becomes

$$\begin{aligned}\min_{B,Z}\quad & \sum_{l=1}^{m} tr([B^l, Z](S^l - W^l)[B^l, Z]^T),\\ \text{s.t.}\quad & B^l \in \{-1,1\}^{r\times n_l}, Z \in \{-1,1\}^{r\times q}\\ & B\mathbf{1} = 0,\\ & BB^T = nI.\end{aligned} \tag{18}$$

Note that the two constraints are related to the whole binary codes. Directly optimizing with the two constraints needs to assemble all the binary codes together [30], which leads to prohibitive communication cost. To optimize the whole problem in a distributed manner, we introduce the auxiliary variables $\{D^l\}, \{M^l\}$, with $D^l = B^l\mathbf{1}_{n_l}$ and $M^l = B^l(B^l)^T$, for all $l \in \{1,\ldots,m\}$. Since

$$B\mathbf{1}_n = [B^1, B^2, \ldots, B^m]\mathbf{1}_n = \sum_{l=1}^{m} B^l\mathbf{1}_{n_l},$$

and

$$BB^T = [B^1, B^2, \ldots, B^m][B^1, B^2, \ldots, B^m]^T = \sum_{l=1}^{m} B^l(B^l)^T,$$

the above problem can be rewritten as

$$\begin{aligned}\min_{B,Z}\quad & \sum_{l=1}^{m} tr([B^l, Z](S^l - W^l)[B^l, Z]^T),\\ \text{s.t.}\quad & B^l \in \{-1,1\}^{r\times n_l}, Z \in \{-1,1\}^{r\times q}\\ & D^l = B^l\mathbf{1}_{n_l}, \forall l \in \{1,\ldots,m\},\\ & M^l = B^l(B^l)^T, \forall l \in \{1,\ldots,m\},\\ & \sum_{l=1}^{m} D^l = 0,\quad \sum_{l=1}^{m} M^l = nI_r.\end{aligned} \tag{19}$$

Like [30], by introducing the penalty terms, we can further rewrite the problem as

$$\begin{aligned}\min_{B^l,Z,D^l,M^l}\quad & \sum_{l=1}^{m} [tr([B^l, Z](S^l - W^l)[B^l, Z]^T)\\ & + \mu||B^l\mathbf{1}_{n_l} - D^l||^2 + \eta||B^l(B^l)^T - M^l||^2]\\ \text{s.t.}\quad & B^l \in \{-1,1\}^{r\times n_l}, Z \in \{-1,1\}^{r\times q},\\ & \sum_{l=1}^{m} D^l = 0,\quad \sum_{l=1}^{m} M^l = nI_r.\end{aligned} \tag{20}$$

One can see that problem (19) is equivalent to problem (20) with sufficiently large μ, η. As shown in [30], [47], the bits balance and uncorrelation constraints in the hashing problem are only used to formulate the desired properties, which are not treated as hard constraints that cannot be violated, so in real applications, we usually set moderate μ, η.

B. Distributed Learning Algorithm

Problem (20) involves a lot of variables and it is difficult to directly optimize the problem with all the variable. For tractability, such a problem is usually solved by alternating optimization [30], [40], namely sequentially optimizing the problem with respect to one set of the variables while keeping the others fixed. In the following, we give the specific optimization procedure step by step.

Optimizing Problem (20) w.r.t. B, Z: First, we optimize the problem (20) with respect to the hash code variables B, Z while keeping the variables $\{D^l\}, \{M^l\}$ fixed, which is shown as follows

$$\begin{aligned}\min_{B^l,Z} \quad & \sum_{l=1}^{m}[tr([B^l,Z](S^l-W^l)[B^l,Z]^T)+\mu||B^l\mathbf{1}_{n_l}-D^l||^2 \\ & +\eta||B^l(B^l)^T-M^l||^2] \\ \text{s.t.} \quad & B^l\in\{-1,1\}^{r\times n_l}, Z\in\{-1,1\}^{r\times q}, \end{aligned} \tag{21}$$

Also, we aim to transform this problem into an equivalent continuous optimization problem and then devise distributed optimization algorithm based on DC programming to solve this problem. Before giving the optimization method, we want to make an uninfluential modification to problem (21), which will bring much convenience for the subsequent optimization. The matrix M^l is not guaranteed to be positive semidefinite. Let ν_l be the smallest eigenvalue of M^l. Define

$$\hat{M}^l=\begin{cases} M^l-\nu I_r, & \text{if} \quad \nu_l<0, \\ M^l, & \text{if} \quad \nu_l\geq 0. \end{cases} \tag{22}$$

Then $\hat{M}^l$ is positive semidefinite. It can be verified that

$$||B^l(B^l)^T-\hat{M}^l||^2=||B^l(B^l)^T-M^l||^2+const,$$

for all $B^l\in\{-1,1\}^{r\times n_l}$. Thus, optimizing problem (21) is equivalent to optimizing

$$\begin{aligned}\min_{B^l,Z} \quad & \sum_{l=1}^{m}[tr([B^l,Z](S^l-W^l)[B^l,Z]^T)+\mu||B^l\mathbf{1}_{n_l}-D^l||^2 \\ & +\eta||B^l(B^l)^T-\hat{M}^l||^2] \\ \text{s.t.} \quad & B^l\in\{-1,1\}^{r\times n_l}, Z\in\{-1,1\}^{r\times q}. \end{aligned} \tag{23}$$

According to the transformation method shown in Section III-B, we can transform the above problem into the following equivalent continuous optimization problem

$$\begin{aligned}\min_{B,Z} \quad & \sum_{l=1}^{m}[tr([B^l,Z](S^l-W^l)[B^l,Z]^T)+\mu||B^l\mathbf{1}_{n_l}-D^l||^2 \\ & +\eta||B^l(B^l)^T-\hat{M}^l||^2+\gamma_l\varphi_l(B^l)+\frac{1}{m}\gamma_z\varphi(Z)], \\ \text{s.t.} \quad & B^l\in[-1,1]^{r\times n_l}, Z\in[-1,1]^{r\times q}. \end{aligned} \tag{24}$$

Then we adopt the method of DC programming to find an optimal solution of problem (24), which is also an optimal solution of (23). Note that

$$\begin{aligned}||B^l(B^l)^T-\hat{M}^l||^2 &= ||B^l(B^l)^T||^2-2tr((\hat{M}^l)^TB^l(B^l)^T) \\ &\quad +||\hat{M}^l||^2,\end{aligned}$$

where $||B^l(B^l)^T||^2, 2tr((\hat{M}^l)^TB^l(B^l)^T)$ are convex and $||M^l||^2$ is a constant. Therefore, we can represent the problem (24) as the following DC programming

$$\begin{aligned}\min_{B,Z} \quad & G_1(E)-H_1(E) \\ \text{s.t.} \quad & E\in[-1,1]^{r\times(n+q)}, \end{aligned} \tag{25}$$

where $E=[B,Z]$ and

$$\begin{aligned}G_1(E) &= \sum_{l=1}^{m}[tr([B^l,Z](S^l-W^l)[B^l,Z]^T) \\ &\quad +\mu||B^l\mathbf{1}_{n_l}-D^l||^2+\eta||B^l(B^l)^T||^2], \\ H_1(E) &= \sum_{l=1}^{m}\Big[2\eta tr((\hat{M}^l)^TB^l(B^l)^T) \\ &\quad -\gamma\varphi_l(B^l)-\frac{1}{m}\gamma_z\varphi(Z)\Big].\end{aligned}$$

It can be verified that $G_1(E), H_1(E)$ are both convex and such a decomposition satisfies the distributed setting. As stated before, DCA solves problem (25) by repeating

$$A=\nabla H_1(E), \tag{26}$$

$$E=\arg\min\nolimits_{E\in[-1,1]^{r\times(n+q)}}\{G_1(E)-tr(E^TA)\}, \tag{27}$$

where $\nabla H_1(E)=[\frac{\partial H_1}{\partial B},\frac{\partial H_1}{\partial Z}]$, and $\frac{\partial H_1}{\partial B}=[\frac{\partial H_1}{\partial B^1},\ldots,\frac{\partial H_1}{\partial B^m}]$. Similar to Section III-C, for simplicity, we also set

$$A^l=\frac{\partial H_1}{\partial B^l}=4\eta(\hat{M}^l)^TB^l+2\gamma_lB^l$$

for all $l\in\{1,\ldots,m\}$, and set

$$A_z=\frac{1}{m}\frac{\partial H_1}{\partial Z}=\frac{2\gamma_z}{m}Z.$$

Then problem (27) can be represented as the following distributed convex optimization problem

$$\begin{aligned}\min_{B,Z} \quad & \sum_{l=1}^{m}[tr([B^l,Z](S^l-W^l)[B^l,Z]^T)+\mu||B^l\mathbf{1}_{n_l}-D^l||^2 \\ & +\eta||B^l(B^l)^T||^2-tr((B^l)^TA^l)-tr(Z^TA_z)] \\ \text{s.t.} \quad & B^l\in[-1,1]^{r\times n_l}, Z\in[-1,1]^{r\times q}, \end{aligned} \tag{28}$$

which can also be solved using the distributed projected gradient descent algorithm. Let

$$\begin{aligned}J_l(B^l,Z^l) &= tr([B^l,Z](S^l-W^l)[B^l,Z]^T) \\ &\quad +\mu||B^l\mathbf{1}_{n_l}-D^l||^2 \\ &\quad +\eta||B^l(B^l)^T||^2-tr((B^l)^TA^l)-tr(Z^TA_z)\end{aligned}$$

The gradient of $J_l(B^l,Z^l)$ with respect to the variable $[B^l,Z^l]$ is

$$\nabla J_l(B^l,Z^l)=2[B^l,Z^l](S^l-W^l)-[A_1^l,A_z^l], \tag{29}$$

with

$$A_1^l=-2\mu(B^l\mathbf{1}_{n_l}-D^l)\mathbf{1}_{n_l}^T-4\eta B^l(B^l)^TB^l+A^l.$$

Algorithm 3: Distributed Projected Gradient Descent (dPGD1) Algorithm for Subproblem (28).

```
Input: $B^l \in \mathbb{R}^{r\times n_l}$, $Z^l \in \mathbb{R}^{r\times q}$, $A^l \in \mathbb{R}^{r\times n_l}$, $A_z^l \in \mathbb{R}^{r\times q}$,
  $D^l \in \mathbb{R}^r$, and $M^l \in \mathbb{R}^{r\times r}$, for all $l \in \{1,\dots,m\}$.
Repeat
    Computing gradient step: each agent $l$ compute the
      gradient $\nabla J_l(B^l, Z^l)$ by equation (29);
    Projected gradient descent step: Each agent $l$ computes
      $[B^l, R^l] = \mathcal{P}_{\mathcal{D}_l}([B^l, Z^l] - \alpha\nabla J_l(B^l, Z^l));$
    Communication step: Each agent $l$ shares $R^l$ with all the
      neighbors $j \in \mathcal{N}_l$ and updates $Z^l$ by
      $Z^l = \sum_{j=1}^{m} w_{lj} R^j;$
Until convergence condition holds.
Output: $B^l \in \mathbb{R}^{r\times n_l}$, $Z^l \in \mathbb{R}^{r\times q}$, for all $l \in \{1,\dots,m\}$.
```

We show the algorithm for solving (28) in Algorithm 3.

Optimizing Problem (20) w.r.t. D, M: We give the detailed derivations of this part in the Appendix.

Algorithm Flow: For clarity, we show the whole algorithm flow for solving the distributed hashing problem (20) in Algorithm 4.

The updates in each step is implemented in parallel on all the agents, and all the updates are finished by agent l independently or with moderate communication with its neighbors, which satisfies the distributed setting. As a consequence, the proposed distributed discrete hashing algorithm works efficiently.

Algorithm 4: dCDH.

```
Input the parameters $\{\gamma_l\}, \gamma_z, \mu, \eta, \rho$ and anchor points
 $\{a_i\}_{i=1}^q$. Each agent $l$ constructs the $W^l$ and $S^l$ using the
 local data points and anchor points. Initialize the variables
 $B^l, Z^l, D^l, \theta_l, M^l, \Phi_l$ and the Lagrangian multiplier $\lambda_l, \Lambda_l$
 for each agent $l$.
Loop until convergence or reach $T$ times:
  step 1: Optimizing $B, Z$ with all the other variables fixed:
  -  Each agent $l$ computes $\hat{M}^l$ according to (22);
    Repeat
    -  Each agent $l$ computes
       $A^l = 4\eta(\hat{M}^l)^T B^l + 2\gamma_l B^l, A_z^l = \frac{2\gamma_z}{m} Z^l;$
    -  Each agent $l$ updates $[B^l, Z^l]$ by
  $\{B^l, Z^l\} = \text{dPGD1}(\{B^l\}, \{Z^l\}, \{A^l\}, \{A_z^l\}, \{D^l\}, \{M^l\}).$
    Until convergence condition holds.
  step 2: Optimizing $\{D^l\}, \{M^l\}$ with all the other
   variables fixed:
  -  Each agent $l$ transmits $\theta_l, \Phi_l$ to its neighbors;
    Repeat
    -  Each agent $l$ updates $\theta_l$ and $\Phi_l$ according to (46) and
  (53);
    -  Each agent $l$ transmits $\theta_l$, $\Phi_l$ to its neighbors;
    -  Each agent $l$ updates $\lambda_l$ and $\Lambda_l$ according to (47) and
  (54);
    Until convergence condition holds.
  -  Each agent $l$ updates $D^l$ by $D^l = \frac{1}{2}(2B^l\mathbf{1} - \theta_l)$ and
   $M^l$ by $M^l = \frac{1}{2}(2B^l(B^l)^T - \Phi_l)$.
  Output The binary codes $\{B^l\}$.
```

V. Out of Sample Extension and Hash Function Learning

The above methods are used to learn binary codes of the training data in a distributed manner. Generally, the hashing methods will construct a hash function for encoding out-of-sample queries. In this paper, for convenience, we adopt the nonlinear hash function $f(x) = \text{sign}(P^T K(x))$, where $K(x) = [\exp(-||x-a_1||^2/\sigma), \dots, \exp(-||x-a_q||^2/\sigma)]^T$ is a q-dimensional column vector obtained using the RBF kernel mapping and $P \in \mathbb{R}^{q\times r}$ is a projection matrix projecting $K(x)$ into the low dimensional space. The vectors $a_1, \dots, a_q$ are the anchor points mentioned above and σ is the kernel width. Like [30], when the binary codes of the training data are obtained, we learn the hash function by solving the following problem

$$\min_P ||B - P^T K(X)||^2. \tag{30}$$

Different from the centralized hashing method in [30], in which such a problem can be directly solved by $P = (K(X)K(X)^T)^{-1}K(X)B^T$, since the data are distributed across different agents in this paper, we also need to learn the hash function in a distributed manner. We reformulate the problem (30) as the following distributed optimization problem

$$\min_P \sum_l^m ||B^l - P^T K(X_l)||^2. \tag{31}$$

We can still use the ADMM method to solve this problem. According to the same ADMM update rule shown in Appendix A, the iterative equations can be formulated as follows

$$\begin{cases} P_l(k+1) & = (2\rho|\mathcal{N}_l|I_p + 2K(X_l)K(X_l)^T)^{-1} \\ & \quad \left(2K(X_l)(B^l)^T - \Pi(k) + 2\rho\sum_{j\in\mathcal{N}_l} P_j(k)\right), \\ \Pi_l(k+1) & = \Pi_l(k) + 2\rho\sum_{j\in\mathcal{N}_l}\left(P_l(k+1) - P_j(k+1)\right), \end{cases} \tag{32}$$

where P_l are local variables and Π_l are Lagrangian multipliers.

VI. Complexity Analysis

In this section, we give the communication and computational complexity analyses of the proposed dDH and dCDH algorithms. Before giving the analyses, we first recall all the related quantities mentioned in this paper: the dimension of the data d, the number of the agents m, the number of the anchor points q, the length of the code r, the number of the data points n_l in agent l. In addition, the size of the truncated similarity matrix U^l used for constructing the local similarity matrix W^l

is $(n_l + q) \times p$. Among all the quantities, n_l should be the only large number, while the others are all much smaller than n_l.

A. Communication Complexity

dDH: In the initialization stage, the network needs to broadcast the anchor points $\{a_i\}_{i=1}^q$. The communication complexity of this step is $\mathcal{O}(qdm)$ and such a step only needs to be implemented once. In the main iteration, to update the variable Z, each agent l needs to transmit R^l to its $|\mathcal{N}_l|$ neighbors and the communication complexity of this step is $\mathcal{O}(rq|\mathcal{N}_l|)$. When constructing the hash function, each agent l needs to transmit its local variable P_l to its $|\mathcal{N}_l|$ neighbors and the communication complexity is $\mathcal{O}(rq|\mathcal{N}_l|)$. Therefore, the overall communication complexity of each agent l is $\mathcal{O}(qdm + rq|\mathcal{N}_l|)$, which is independent of the data size n_l.

dCDH: In the initialization stage, the communication complexity of this step is also $\mathcal{O}(qdm)$. In step 1, to update the variable Z, each agent l also needs to transmit R^l to its $|\mathcal{N}_l|$ neighbors and the communication complexity of this step is $\mathcal{O}(rq|\mathcal{N}_l|)$. In step 2, each agent l needs to transmit its local variable θ_l and Φ_l to its $|\mathcal{N}_l|$ neighbors and the communication complexity of this step is $\mathcal{O}(r|\mathcal{N}_l| + r^2|\mathcal{N}_l|)$. In the stage of constructing the hash function, the communication complexity is also $\mathcal{O}(rq|\mathcal{N}_l|)$. Therefore, the overall communication complexity of each agent l is $\mathcal{O}(qdm + r|\mathcal{N}_l|(r + q))$, which is independent of the data size n_l.

B. Computation Complexity

dDH: In the initialization stage, for each agent l, the time complexity of constructing $K(X_l)$ is $\mathcal{O}(dqn_l)$. The time complexity of constructing W^l using the method in [7] is $\mathcal{O}(pd(n_l + q))$. In the main iteration, the time complexity of updating B and Z by dPGD algorithm is $\mathcal{O}(rp(n_l + q)t)$, where t is the iteration number of dPGD algorithm. In the stage of constructing the hash function, the time complexity is $\mathcal{O}(q(q + r)n_l + q^{2.376})$. In summary, the main time complexity of each agent l is $\mathcal{O}((dp + dq + prt + q^2 + qr)n_l)$, which is linear of the data size n_l.

dCDH: In the initialization stage, for each agent l, the time complexity of constructing $K(X_l)$ is $\mathcal{O}(dqn_l)$ and the time complexity of constructing W^l is $\mathcal{O}(pd(n_l + q))$. In step 1, to obtain $\hat{M}^l$, each agent l needs to compute the smallest eigenvalue of M^l. The time complexity of this step is $\mathcal{O}(r^{2.376})$ using the Coppersmith and Winograd algorithm [41]. The time complexity of computing A_l is $\mathcal{O}(r^2 n_l)$ and the time complexity of updating B and Z by dPGD1 algorithm is $\mathcal{O}((rp(n_l + q) + r^2 n_l)t')$, where t' is the iteration number of dPGD1 algorithm. In step 2, the time complexity of updating Φ_l is $\mathcal{O}(r^2 n_l)$. In the stage of constructing the hash function, the time complexity is $\mathcal{O}(q(q + r)n_l + q^{2.376})$. In summary, the overall time complexity of each agent l is $\mathcal{O}((dp + dq + prt' + q^2 + qr + r^2 t')n_l)$, which is linear of the data size n_l.

VII. Experiments

In this section, we test the proposed distributed hashing algorithms by image retrieval experiments. In all the experiments below, we assume that the data are distributed across a connected network consisting of 10 agents. The benchmarks used for test are: CIFAR-10 [13], MNIST [1], and NUS-WIDE [7]. We empirically set the parameters $\gamma_l = \gamma_z = \gamma = 1, \mu = 1e - 3$, $\eta = 1e - 4$. In the following, we will investigate the impacts of these parameters. We set the maximum iteration $T = 10$. The performance of the algorithm is not sensitive to the stepsize ρ. The stepsize ρ mainly affects the convergence speed of the ADMM based algorithm and generally a small stepsize gives faster convergence. In the following experiments, we set $\rho = 0.1$.

In the proposed algorithms, we need the anchor points to construct W^l. Generally, the more anchor points, the better performance the hashing method is expected to achieve. However, the anchor points will bring additional computation and communication, while a moderate number of the anchor points are usually enough to ensure the performance. In real applications, one can choose 500-5000 anchor points according to the allowable computing resource. In this section, we choose 1000 anchor points for all the examples. We compare the proposed dDH and dCDH algorithms with the state-of-the-art centralized unsupervised hashing methods including AGH [7], ITQ [8], SGH [46], OCH [31], and DPLM [30]. Among them, DPLM is a discrete hashing method, which learns binary codes without relaxations, and the other four methods are all relaxation hashing methods. In addition, we also compare the proposed algorithms with the existing distributed unsupervised hashing methods including DisH [26], SDH [28], PDH [28]. Among the hashing methods for comparison, AGH, SGH, SDH, PDH also need to use the anchor points. We also choose 1000 anchor points for these methods. In the following examples, for the distributed hashing methods, we divide the training data points into 10 splits evenly and each agent collects one split. For the compared centralized algorithms, we use a single agent to collect all the training data points for learning.

All the results in the following experiments are averaged over 50 independent runs.

A. CIFAR-10: Test on Tiny Natural Images

In this example, the dataset for test is CIFAR-10. CIFAR-10 consists of 60 K images in 10 classes and each class contains 6 K images. Each image is represented by a 512-dimensional GIST descriptor extracted from a 32×32 color image. We randomly select 1000 images from the dataset as the query (test) set and use the remaining data as the training set. Since each data point in CIFAR-10 is assigned a class label, the ground truth is determined based on label agreement.

The results in terms of mean average precision (MAP) and precision of top 5000 returns are shown in Table I. From Table I, we see that the proposed dDH and dCDH methods outperform all the other distributed hashing algorithms and most of the centralized hashing methods except the discrete hashing method DPLM, which demonstrates the effectiveness and efficiency of the proposed algorithms. Note that the performance of dDH and dCDH is very close to that of DPLM, which is the best competitor in the centralized methods, so the proposed distributed hashing algorithms do not lose much quality compared with the

TABLE I
MEAN AVERAGE PRECISION (MAP) AND PRECISION OF TOP 5000 RETURNS ON CIFAR-10

Methods	MAP				Precision@5000			
	48bits	64its	96bits	128bits	48bits	64its	96bits	128bits
AGH	0.1476	0.1442	0.1406	0.1382	0.1728	0.1716	0.1699	0.1658
SGH	0.1647	0.1749	0.1720	0.1787	0.1820	0.1850	0.1891	0.1934
ITQ	0.1632	0.1673	0.1747	0.1749	0.1926	0.1961	0.2013	0.2022
OCH	0.1675	0.1702	0.1740	0.1769	0.1921	0.1981	0.2010	0.2039
DPLM	**0.1781**	**0.1895**	**0.1886**	**0.1889**	**0.2013**	**0.2111**	**0.2113**	**0.2152**
SDH	0.1750	0.1738	0.1794	0.1804	0.1963	0.1944	0.1967	0.2018
PDH	0.1747	0.1759	0.1786	0.1816	0.1947	0.1952	0.1977	0.1994
DisH	0.1721	0.1713	0.1732	0.1763	0.1965	0.1966	0.1974	0.1989
dDH	**0.1800**	**0.1860**	**0.1877**	**0.1884**	**0.2011**	**0.2082**	**0.2089**	**0.2094**
dCDH	**0.1830**	**0.1882**	**0.1881**	**0.1902**	**0.2030**	**0.2087**	**0.2104**	**0.2144**

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

Methods	Training time (s)			
	48bits	64its	96bits	128bits
AGH	21.2577	21.2343	22.1464	22.1648
SGH	46.2377	64.5533	90.367	112.6654
ITQ	2.2243	3.6758	4.3156	5.4427
OCH	9.2542	10.4875	14.6657	17.7145
DPLM	3.6757	4.3695	6.7570	8.2106
SDH	4.8516	5.5364	6.7721	8.9681
PDH	1.9962	2.7747	2.8987	3.2142
DisH	2.1936	2.4576	2.8554	3.0164
dDH	2.3366	2.7857	3.1042	3.4588
dCDH	7.4023	8.2359	10.1486	12.0571

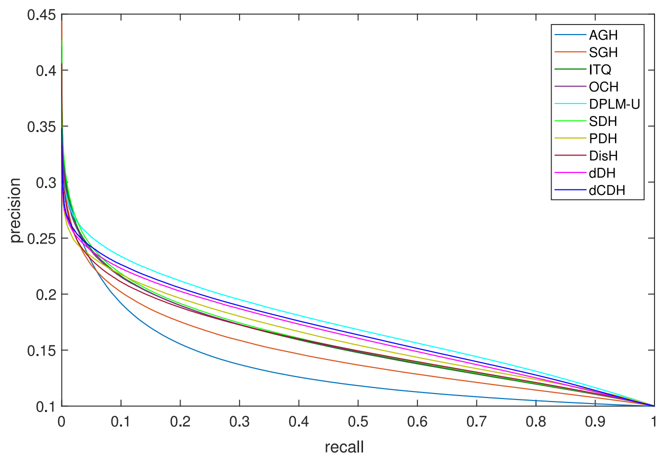


Fig. 1. The precision-recall curves with 128 bits on CIFAR-10.

centralized discrete hashing method. In addition, we see that dCDH achieves better results than dDH, which implies that the bits uncorrelation and balance constraints can improve the quality of the binary codes.

Then we further report the training time consumption in Table II. The results in Table II show that the proposed algorithms are time efficient. Though dCDH takes the longer time than the other distributed hashing methods, it achieves the best accuracy performance.

We also report the complete precision-recall curves with 128 bits code in Fig. 1. The results shown in Fig. 1 are consistent to those given in Table I. The curves of dDH and dCDH are close to the curve of DPLM, and outperform all the other competitors.

Next, we investigate the impact of the parameter γ. We plot the curve of the quantization error with respect to γ with 64 bits using dDH and dCDH algorithms in Fig. 2. We see that the quantization error decreases as γ increases, and when $\gamma \geq 1$, the quantization error is close to zero. Then we further show the MAP results with respect to γ in Fig. 3. We see that though larger γ can guarantee smaller quantization loss, the performance of the algorithms become worse on the contrary when γ is very large. That is because the penalty term will dominate the overall loss with large γ, making the optimization difficult. So we should use a moderate γ to balance the quantization error and the optimization difficulty.

We also investigate the sensitivity of the parameters μ and η. The MAP results of dCDH with 64 bits with respect to different values of μ and η are shown in Fig. 4. From the figures, we see that the retrieval accuracy improves as μ, η increase, but decreases when μ, η are large, which implies that the bit uncorrelation and balance constraints can make the binary codes more efficient, but excessively emphasizing the two constraints will reduce the quality of the binary codes on the contrary. These results are in line with the results shown in [47].

B. MNIST: Test on Hand-Written Digits

In this example, we test the above hashing methods on the benchmark dataset MNIST. MNIST consists of 70000 images of handwritten digits from 0 to 9. All the images in the dataset are represented by 784-dimensional vectors. Also, the ground truth neighbors are determined by label agreement.

Results in terms of MAP and precision of top 5000 returns are shown in Table III. From Table III, we see that the performance of the proposed dDH and dCDH algorithm is still close to that of DPLM, and is better than the performance of all the other methods, which is consistent to the results shown in last example.

C. NUS-WIDE: Test on Large-Scale Dataset

We further test all the hashing methods on the large-scale dataset NUS-WIDE. NUS-WIDE consists of around 270,000 web images associated with 81 ground truth concept labels. Each image in NUS-WIDE is represented by a 500-dimensional Bag-of-Words (BOW) feature. Each two data points are defined as neighbors if they share at least one label. We collect 21 most frequent labels and randomly select 100 images for each label as queries. The remaining images are used for training. Results

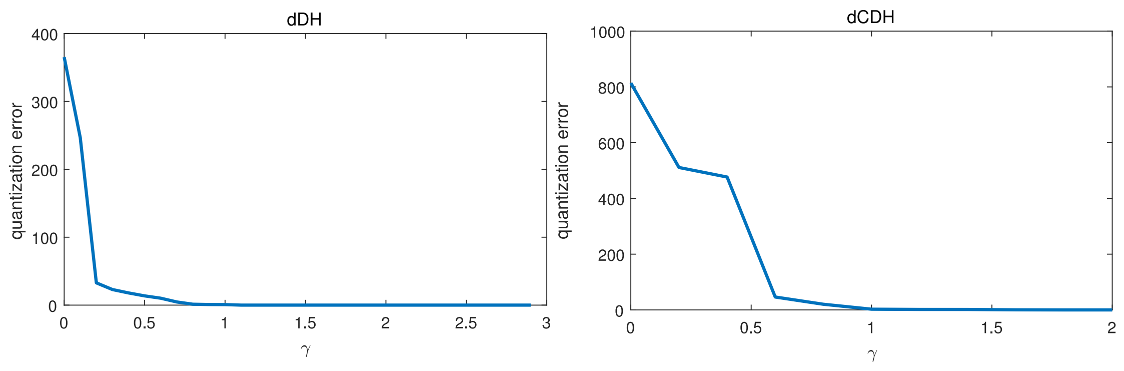


Fig. 2. The impact of γ on the quantization error with 64 bits.

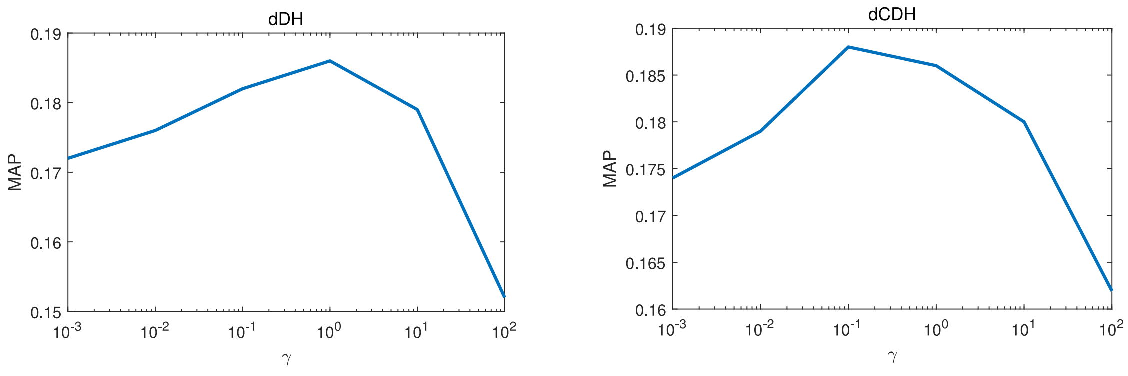


Fig. 3. MAP results w.r.t. γ with 64 bits on CIFAR-10.

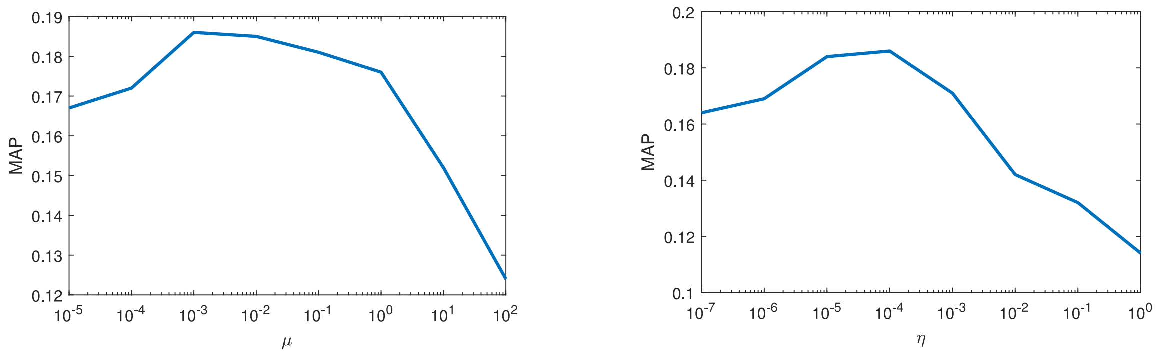


Fig. 4. MAP results w.r.t. μ, ρ with 64 bits on CIFAR-10.

TABLE III
MEAN AVERAGE PRECISION (MAP) AND PRECISION OF TOP 5000 RETURNS ON MNIST

Methods	MAP				Precision@5000			
	48bits	64its	96bits	128bits	48bits	64its	96bits	128bits
AGH	0.3136	0.2950	0.2703	0.2533	0.3857	0.3631	0.3322	0.3117
SGH	0.3968	0.4096	0.4198	0.4258	0.4499	0.4615	0.4709	0.4755
ITQ	0.4043	0.4257	0.4368	0.4449	0.4546	0.4749	0.4863	0.4929
OCH	0.3864	0.4136	0.4303	0.4275	0.4405	0.4658	0.4790	0.4783
DPLM	0.5019	**0.5425**	**0.6252**	**0.5928**	**0.5553**	0.5779	**0.6424**	0.6218
SDH	0.3159	0.3281	0.3473	0.3751	0.3553	0.3750	0.3882	0.4195
PDH	0.3144	0.3267	0.3482	0.3695	0.3521	0.3714	0.3894	0.4203
DisH	0.3683	0.3862	0.4002	0.4149	0.4185	0.4339	0.4490	0.4626
dDH	**0.5129**	**0.5560**	**0.5565**	**0.6081**	**0.5563**	**0.6014**	**0.5982**	**0.6501**
dCDH	**0.5157**	**0.5794**	**0.5893**	**0.6313**	**0.5630**	**0.6234**	**0.6402**	**0.6791**

TABLE IV
MEAN AVERAGE PRECISION (MAP) AND PRECISION OF TOP 5000 RETURNS ON NUS-WIDE

Methods	MAP				Precision@5000			
	48bits	64its	96bits	128bits	48bits	64its	96bits	128bits
AGH	0.2635	0.2645	0.2658	0.2667	0.3107	0.3140	0.3199	0.3234
SGH	0.2589	0.2600	0.2613	0.2617	0.3134	0.3089	0.3051	0.3048
ITQ	0.2673	0.2702	0.2716	0.2721	0.3331	0.3352	0.3375	0.3411
OCH	0.2659	0.2717	0.2731	0.2734	0.3342	**0.3398**	0.3404	**0.3442**
DPLM	**0.2788**	**0.2827**	**0.2829**	**0.2826**	**0.3365**	0.3394	**0.3464**	0.3430
SDH	0.2654	0.2630	0.2652	0.2648	0.3177	0.3212	0.3197	0.3189
PDH	0.2625	0.2674	0.2677	0.2675	0.3129	0.3224	0.3184	0.3201
DisH	0.2597	0.2671	0.2699	0.2703	0.3106	0.3167	0.3178	0.3249
dDH	**0.2707**	**0.2737**	**0.2774**	**0.2780**	**0.3237**	**0.3317**	**0.3390**	**0.3428**
dCDH	**0.2742**	**0.2754**	**0.2785**	**0.2796**	**0.3271**	**0.3364**	**0.3410**	**0.3442**

in terms of MAP and precision of top 5000 returns are shown in Table IV. We see that the proposed dDH and dCDH algorithms perform better than the other distributed hashing methods, and the performance of the proposed algorithms is close to that of the best centralized hashing method.

VIII. CONCLUSION

In this paper, we proposed a distributed discrete hashing method to learn binary codes of distributed data. To guarantee the effectiveness of the binary codes, we devised a distributed discrete hashing algorithm, which solves the hashing problem without using any relaxations. To make the binary codes more efficient, we further added the bits balance and uncorrelation constraints, and we represented the hashing problem with the bits balance and uncorrelation constraints as a tractable distributed optimization problem. Then we also proposed a distributed constrained discrete hashing algorithm for solving this problem. Experiments on large-scale benchmark datasets were provided to show the superiorities of the proposed algorithms.

APPENDIX A
OPTIMIZING PROBLEM (20) W.R.T. D, M

With all the variables fixed but $\{D^l\}, \{M^l\}$, the problem is formulated as

$$\min_{D^l, M^l} \sum_{l=1}^{m} [||B^l \mathbf{1}_{n_l} - D^l||^2 + ||B^l (B^l)^T - M^l||^2]$$
$$\text{s.t.} \quad \sum_{l=1}^{m} D^l = 0, \quad \sum_{l=1}^{m} M^l = nI_r. \tag{33}$$

The above problem can be separately solved by solving

$$\min_{D^l} \sum_{l=1}^{m} ||B^l \mathbf{1}_{n_l} - D^l||^2$$
$$\text{s.t.} \quad \sum_{l=1}^{m} D^l = 0, \tag{34}$$

and

$$\min_{M^l} \sum_{l=1}^{m} ||B^l (B^l)^T - M^l||^2$$
$$\text{s.t.} \quad \sum_{l=1}^{m} M^l = nI_r. \tag{35}$$

We first solve problem (34). This should be the first time that such a problem arises in distributed hashing, since the existing distributed hashing methods either did not consider the bits balance and uncorrelation constraints or relaxed them for tractability. In the literature, most of the distributed optimization tasks can be cast as an optimization problem of the following form

$$\min_{x} \sum_{l=1}^{m} f_l(x)$$
$$\text{s.t.} \quad x \in \mathcal{X}, \tag{36}$$

in which all the local objective functions $\{f_l\}$ have the same optimized variable. However, in (33), each agent l owns an individual variable D^l. The variables $\{D^l\}$ are different but coupled together through the linear constraint. In this situation, it is difficult to directly apply the existing standard distributed algorithms, such as distributed gradient descent [50] and distributed ADMM [49], to solving this problem.

Since problem (33) is a convex optimization problem, we can solve the problem by solving its Lagrangian dual problem [52]. The Lagrangian dual problem of (33) can be formulated as

$$\min_{\theta \in \mathbb{R}^r} \sum_{l=1}^{m} \phi_l(\theta), \tag{37}$$

where

$$\phi_l(\theta) = \max_{D^l} \{-||B^l \mathbf{1}_{n_l} - D^l||^2 - (D^l)^T \theta\}, \tag{38}$$

and θ is the dual variable. Note that the optimization problem in (38) can be analytically solved by

$$D^l = \frac{1}{2}(2B^l \mathbf{1} - \theta). \tag{39}$$

Then we have that

$$\phi_l(\theta) = \frac{1}{4}||\theta||^2 - (B^l \mathbf{1}_{n_l})^T \theta, \tag{40}$$

and problem (37) can be represented as

$$\min_{\theta \in \mathbb{R}^r} \sum_{l=1}^{m} \left[\frac{1}{4}||\theta||^2 - (B^l \mathbf{1}_{n_l})^T \theta \right], \tag{41}$$

which is the standard form of the distributed optimization as shown in (36). It is easy to verify that problem (33) has zero duality gap [53]. Therefore, we can first solve the dual problem

(41), and then each agent l can obtain the corresponding optimal solution D^l by (39).

To solve problem (41), we can devise distributed algorithms based on the idea of ADMM. To apply ADMM, first, we need to rewrite the problem (41) as follows

$$\begin{aligned}\min_{\theta_l \in \mathbb{R}^r} \quad & \sum_{l=1}^{m} \left[\frac{1}{4}||\theta_l||^2 - (B^l \mathbf{1}_{n_l})^T \theta_l\right], \\ \text{s.t.} \quad & \theta_l = \theta_j, \quad \forall j \in \mathcal{N}_l, \forall l \in \{1, \ldots, m\}, \qquad (42)\end{aligned}$$

where $\theta_l \in \mathbb{R}^r$ are local variables, and $\theta_l = \theta_j$ are consensus constraints. Because of the transitivity between neighboring nodes in a connected network, we can only consider the consensus constraints between local neighbors rather than all the agents. The augmented Lagrangian function of (42) is formulated as

$$\sum_{l=1}^{m} \left[\frac{1}{4}||\theta_l||^2 - (B^l \mathbf{1}_{n_l})^T \theta_l + \sum_{j \in \mathcal{N}_l} \lambda_{l,j}^T (\theta_l - \theta_j) + \frac{\rho}{2} \sum_{j \in \mathcal{N}_l} ||\theta_l - \theta_j||^2\right], \qquad (43)$$

where $\lambda_{l,j}$ are Lagrangian multipliers and ρ is the penalty parameter of augmented Lagrangian. ADMM solves such a problem by repeating the following two steps [26], [27]:

$$\begin{cases} \theta_l(k+1) := \arg\min_{\theta_l} \sum_{l=1}^{m} [\frac{1}{4}||\theta_l||^2 - (B^l \mathbf{1}_{n_l})^T \theta_l \\ \quad + \sum_{l=1}^{m} \sum_{j \in \mathcal{N}_l} \lambda_{l,j}^T (k) (\theta_l - \theta_j) \\ \quad + \frac{\rho}{2} \sum_{l=1}^{m} \sum_{j \in \mathcal{N}_l} ||\theta_l - \theta_j||^2, \\ \lambda_{l,j}(k+1) := \lambda_{l,j}(k) + \rho\left(\theta_l(k+1) - \theta_j(k+1)\right), \end{cases} \qquad (44)$$

$\forall l \in \{1, \ldots, m\}, \forall j \in \mathcal{N}_l$, with $\theta_j = \theta_j(k)$ for all $j \neq l$. Neglecting the terms that are independent of θ_l, to update $\theta_l(k+1)$, agent l needs to solve the following problem

$$\min_{\theta_l} \frac{1}{4}||\theta_l||^2 - (B^l \mathbf{1}_{n_l})^T \theta_l + \sum_{j \in \mathcal{N}_l} (\lambda_{l,j}^T \theta_l - \lambda_{j,l}^T \theta_l) + \rho \sum_{j \in \mathcal{N}_l} ||\theta_l - \theta_j(k)||^2. \qquad (45)$$

The above problem involves the term $\lambda_{j,l}^T \theta_l$ because we consider the network is symmetric, namely $l \in \mathcal{N}_j$ if $j \in \mathcal{N}_l$. The above problem can be analytically solved by setting the derivative with respect to θ_l to zero. Then we can get the solution

$$\theta_l(k+1) = \frac{B^l \mathbf{1}_{n_l} - \sum_{j \in \mathcal{N}_l} (\lambda_{l,j}(k) - \lambda_{j,l}(k)) + 2\rho \sum_{j \in \mathcal{N}_l} \theta_j(k)}{2\rho|\mathcal{N}_l| + \frac{1}{2}}.$$

The update iteration can be further simplified. Define the Lagrange multipliers $\lambda_l = \sum_{j \in \mathcal{N}_l} (\lambda_{l,j} - \lambda_{j,l})$, for all $l \in \{1, \ldots, m\}$. Then the whole update procedure can be simplified as

$$\theta_l(k+1) = \frac{B^l \mathbf{1}_{n_l} - \lambda_l(k) + 2\rho \sum_{j \in \mathcal{N}_l} \theta_j(k)}{2\rho|\mathcal{N}_l| + \frac{1}{2}}, \qquad (46)$$

$$\lambda_l(k+1) = \lambda_l(k) + 2\rho \sum_{j \in \mathcal{N}_l} \left(\theta_l(k+1) - \theta_j(k+1)\right), \qquad (47)$$

for all $l \in \{1, \ldots, m\}$.

Then we use the same method to solve (35). The dual problem of (35) can be formulated as

$$\min_{\Phi \in \mathbb{R}^r} \quad \sum_{l=1}^{m} \psi_l(\Phi), \qquad (48)$$

where

$$\psi_l(\Phi) = \max_{M^l} \{-||B^l (B^l)^T - M^l||^2 - tr((M^l)^T \Phi) + \frac{n}{m} tr(\Phi)\}, \qquad (49)$$

and $\Phi \in \mathbb{R}^{r \times r}$ is the dual variable. The optimization problem in (49) has the analytical solution

$$M^l = \frac{1}{2}(2B^l (B^l)^T - \Phi), \qquad (50)$$

and $\psi_l(\Phi)$ can then be represented as

$$\psi_l(\Phi) = \frac{\Phi^2}{4} - tr(B^l (B^l)^T \Phi) + \frac{n}{m} tr(\Phi). \qquad (51)$$

We also use ADMM method to solve the distributed optimization problem

$$\min_{\Phi \in \mathbb{R}^{r \times r}} \quad \sum_{l=1}^{m} \left[\frac{\Phi^2}{4} - tr(B^l (B^l)^T \Phi) + \frac{n}{m} tr(\Phi)\right]. \qquad (52)$$

Since problem (52) are quite similar to problem (41), and we use the same method to solve the problem, to avoid repetition, we directly give the update equations, as follows:

$$\Phi_l(k+1) = \frac{B^l (B^l)^T - \frac{n}{m} I_r - \Lambda_l(k) + 2\rho \sum_{j \in \mathcal{N}_l} \Phi_j(k)}{2\rho|\mathcal{N}_l| + \frac{1}{2}}, \qquad (53)$$

$$\Lambda_l(k+1) = \Lambda_l(k) + 2\rho \sum_{j \in \mathcal{N}_l} \left(\Phi_l(k+1) - \Phi_j(k+1)\right), \qquad (54)$$

for all $l \in \{1, \ldots, m\}$, where Φ_l are local variables and Λ_l are Lagrangian multipliers. After solving the problem (52), each agent l can obtain the corresponding optimal solution M^l by (50).

Appendix B
Distributed Supervised Discrete Hashing

In the above sections, we proposed a distributed unsupervised discrete hashing method. The proposed distributed hashing scheme can also be used for supervised hashing, as long as we replace the unsupervised hashing objective in (18) with a supervised hashing objective. In this section, we adopt the objective of the supervised hashing in [40] as an example. In [40], the bits balance and uncorrelation constraints were not considered. To obtain more efficient binary codes, we add the two constraints, and the distributed supervised hashing problem can be formulated as

$$\begin{aligned}\min_{B^l,W_l} \quad & \sum_{l=1}^{m}[||Y_l - W_l^T B^l||^2 + \tau||W_l|^2] \\ \text{s.t.} \quad & B^l \in \{-1,1\}^{r\times n_l}, \\ & W_l = W_j, \quad \forall j \in \mathcal{N}_l, \forall l \in \{1,\dots,m\}, \\ & B\mathbf{1}_n = 0, \\ & BB^T = nI_r. \end{aligned} \tag{55}$$

where $W_l \in \mathbb{R}^{r\times q}$ are classification matrices, and τ is a parameter. The matrix $Y_l \in \mathbb{R}^{q\times n_l}$ stores the labels of the training data X_l in agent l, and the (i, j)-th entry $Y_l^{i,j} = 1$ if x_j belongs to the i-th class and $Y_l^{i,j} = 0$ otherwise.

Like (19), we can represent the problem as follows

$$\begin{aligned}\min_{B^l,W_l,D^l,M^l} \quad & \sum_{l=1}^{m}[||Y_l - W_l^T B^l||^2 + \tau||W_l|^2 + \mu||B^l\mathbf{1}_{n_l} \\ & - D^l||^2 + \eta||B^l(B^l)^T - M^l||^2] \\ \text{s.t.} \quad & B^l \in \{-1,1\}^{r\times n_l}, \\ & W_l = W_j, \quad \forall j \in \mathcal{N}_l, \forall l \in \{1,\dots,m\}, \\ & \sum_{l=1}^{m} D^l = 0, \quad \sum_{l=1}^{m} M^l = nI_r, \end{aligned} \tag{56}$$

by introducing the auxiliary variables D^l and M^l. Then we can also solve the problem using the alternating optimization method. The optimization problem with respect to the variables $\{D^l\}, \{M^l\}$ is the same as that in the unsupervised case shown above. The optimization problem with respect to the variables $\{B^l\}$ becomes

$$\begin{aligned}\min_{B^l} \quad & \sum_{l=1}^{m}[||Y_l - W_l^T B^l||^2 + \mu||B^l\mathbf{1}_{n_l} - D^l||^2 \\ & + \eta||B^l(B^l)^T - M^l||^2] \\ \text{s.t.} \quad & B^l \in \{-1,1\}^{r\times n_l}. \end{aligned} \tag{57}$$

This problem additionally involves the variables $\{W^l\}$ and the optimization problem with respect to the variables $\{W^l\}$ is

$$\begin{aligned}\min_{W_l} \quad & \sum_{l=1}^{m}[||Y_l - W_l^T B^l||^2 + \tau||W_l|^2] \\ & W_l = W_j, \quad \forall j \in \mathcal{N}_l, \forall l \in \{1,\dots,m\}. \end{aligned} \tag{58}$$

Problem (57) can also be solved by transforming it into an equivalent distributed continuous optimization problem, and problem (58) can be solved using the ADMM method. To avoid repetition, we do not reformulate the specific optimization procedure here.

References

[1] 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.

[2] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan, "Supervised hashing for image retrieval via image representation learning," in *Proc. AAAI Conf. Artif. Intell.*, 2014, vol. 1, pp. 2156–2162.

[3] J. Lu, V. E. Liong, and J. Zhou, "Cost-sensitive local binary feature learning for facial age estimation," *IEEE Trans. Image Process.*, vol. 24, no. 12, pp. 5356–5368, Dec. 2015.

[4] L. Zhang, H. Lu, D. Du, and L. Liu, "Sparse hashing tracking," *IEEE Trans. Image Process.*, vol. 25, no. 2, pp. 840–849, Feb. 2016.

[5] X. Li, G. Lin, C. Shen, A. van den Hengel, and A. Dick, "Learning hash functions using column generation," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 142–150.

[6] 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.

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

[8] 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.

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

[10] 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.

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

[12] W. Liu, J. Wang, R. Ji, and Y. Jiang, "Supervised hashing with kernels," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2012, pp. 2074–2081.

[13] F. Shen, C. Shen, W. Liu, and H. T. Shen, "Supervised discrete hashing," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2015, pp. 37–45.

[14] 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.

[15] 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.

[16] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni, "Locality sensitive hashing scheme based on p-stable distributions," in *Proc. Annu. ACM Symp. Comput. Geometry*, 2004, pp. 253–262.

[17] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *Commun. ACM*, 2008.

[18] W. Liu, J. Wang, S. Kumar, and S. Chang, "Discrete graph hashing," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 3419–3427.

[19] M. Norouzi and D. Blei, "Minimal loss hashing for compact binary codes," in *Proc. 28th Int. Conf. Mach. Learn.,* 2011, pp. 353–360.

[20] T.-T. Do, A.-D. Doan, and N.-M. Cheung, "Learning to hash with binary deep neural network," in *ECCV*, Cham: Springer, 2016, pp. 219–234.

[21] M. A. Carreira-Perpinan and R. Raziperchikolaei, "Hashing with binary autoencoders," in *Proc. IEEE Conf. Comp. Vision Pattern Recognit.*, 2015, pp. 557–566.

[22] K. G. Dizaji, F. Zheng, N. S. Nourabadi, Y. Yang, C. Deng, and H. Huang, "Unsupervised deep generative adversarial hashing network," in *Proc. IEEE Conf. Comp. Vision Pattern Recognit.* 2018, pp. 3664–3673.

[23] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proc. Int. Conf. Very Large Data Bases*, San Francisco, CA, 1999, vol. 99, no. 6, pp. 518–529.

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

[25] S. Wang and C. Li, "Distributed robust optimization in networked system," *IEEE Trans. Cybern.*, vol. 47, no. 8, pp. 2321–2333, Aug. 2017.

[26] 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.

[27] J. Liang, M. Zhang, X. Zeng, and G. Yu, "Distributed dictionary learning for sparse representation in sensor networks," *IEEE Trans. Image Process.*, vol. 23, no. 6, pp. 2528–2541, Jun. 2014.

[28] S. Wang, C. Li, and H. L. Shen, "Distributed graph hashing," *IEEE Trans. Cybern.*, vol. 49, no. 5, pp. 1896–1908, May 2019.

[29] W. Kang, W. Li, and Z. Zhou, "Column sampling based discrete supervised hashing," in *Proc. AAAI Conf. Artif. Intell.*, 2016, pp. 1230–1236.

[30] F. Shen, X. Zhou, Y. Yang, J. Song, H. T. Shen, and D. Tao, "A fast optimization method for general binary code learning," *IEEE Trans. Image Process.*, vol. 25, no. 12, pp. 5610–5621, Dec. 2016.

[31] H. Liu, R. Ji, Y. Wu, and F. Huang, "Ordinal constrained binary code learning for nearest neighbor search," in *Proc. Assoc. Adv. Artif. Intell.*, 2017, pp. 2238–2244.

[32] P. Zhang, W. Zhang, W.-J. Li, and M. Guo, "Supervised hashing with latent factor models," in *Proc. ACM Conf. Inf. Retrieval*, 2014, pp. 173–182.

[33] J. C. Corbett *et al.*, "Spanner: Googles globally-distributed database," in *Proc. 10th USENIX Conf. Oper. Syst. Des. Implementation*, 2012, pp. 251–264.

[34] S. Greenhill and S. Venkatesh, "Distributed query processing for mobile surveillance," in *Proc. 15th ACM Int. Conf. Multimedia*, 2007, pp. 413–422.

[35] A. Howard, M. J. Mataric, and G. S. Sukhatme, "Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem," in *Proc. 5th Int. Symp. Distrib. Auton. Robot. Systs.*, 2002, pp. 299–308.

[36] X. Li, B. Veeravalli, and 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. Tsang, and X. Li, "Spectral embedded hashing for scalable image retrieval," *IEEE Trans. Cybern.*, vol. 44, no. 7, pp. 1180–1190, Jul. 2014.

[38] T. A. Letsche and M. W. Berry, "Large-scale information retrieval with latent semantic indexing," *Inf. Sci.*, vol. 100, no. 1–4, pp. 105–137, 1997.

[39] Q. Chen, B. Lang, X. Liu, and Z. Gu, "DisITQ: A distributed iterative quantization hashing learning algorithm," in *Proc. 9th Int. Symp. Comput. Intell. Des.*, 2016, vol. 2, no. 1–4, pp. 118–123.

[40] D. Zhai, X. Liu, X. Ji, D. Zhao, S. Satoh, and W. Gao, "Supervised distributed hashing for large-scale multimedia retrieval," *IEEE Trans. Multimedia*, vol. 20, no. 3, pp. 675–686, Mar. 2018.

[41] D. Coppersmith and S. Winograd, "On the asymptotic complexity of matrix multiplication," *SIAM J. Comput.*, vol. 11, no. 3, pp. 472–492, 1982.

[42] H. A. L. Thi, T. P. Dinh, and V. N. Huynh, "Exact penalty techniques in DC programming," *J. Global Optim.*, vol. 52, no. 3, pp. 509–535, 2012.

[43] P. Tao, "The DC (difference of convex functions) programming and DCA revisited with DC models of real world nonconvex optimization problems," *Ann. Oper. Res.*, vol. 133, no. 1–4, pp. 23–46, 2005.

[44] P. Tao and L. An, "Convex analysis approach to DC programming: Theory, algorithms and applications," *Acta Math. Vietnamica*, vol. 22, no. 1, pp. 289–355, 1997.

[45] L. An and P. Tao, "Large scale molecular optimization from distances matrices by a DC optimization approach," *SIAM J. Optim.*, vol. 14, no. 1, pp. 77–116, 2003.

[46] Q. Jiang and W. Li, "Scalable graph hashing with feature transformation," in *Proc. Int. Joint Conf. Artif. Intell.*, 2015, pp. 2248–2254.

[47] S. Wang, C. Li, and H L. Shen, "Equivalent continuous formulation of general hashing problem," *IEEE Trans. Cybern.*, to be published, doi: 10.1109/TCYB.2019.2894020.

[48] S. Boyd, L. Xiao, and A. Mutapcic, "Subgradient methods," Lecture Notes of EE3920, Stanford University, Autumn Quarter, Serra Mall, Stanford, CA, USA, 2003-2004.

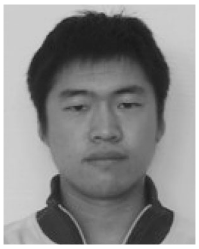
[49] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, 2011.

[50] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multiagent optimization," *IEEE Trans. Autom. Control*, vol. 54, no. 1, pp. 48–61, Jan. 2009.

[51] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," *Syst. Control Lett.*, vol. 53, no. 1, pp. 65–78, 2004.

[52] T. Chang, M. Hong, and X. Wang, "Multi-Agent distributed optimization via inexact consensus ADMM," *IEEE Trans. Signal Process.*, vol. 63, no. 2, pp. 482–497, Jan. 2015.

[53] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.



Shengnan Wang received the B.S. degree in information science and electronic engineering and the Ph.D degree in electronic engineering from Zhejiang University, Hangzhou, China, in 2014 and 2019, respectively. He is a currently a Researcher in Machine Intelligence Technology, Alibaba DAMO Academy. His current research interests include machine learning and optimization.

Chunguang Li (Senior Member, IEEE) 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, and quantitative finance.

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