Abstract—Hashing-based approximate nearest neighbors search has attracted broad research interest, due to its low computational cost and fast retrieval speed. The hashing technique maps the data points into binary codes and, meanwhile, preserves the similarity in the original space. Generally, we need to solve a discrete optimization problem to learn the binary codes and hash functions, which is NP-hard. In the literature, most hashing methods choose to solve a relaxed problem by discarding the discrete constraints. However, such a relaxation scheme will cause large quantization error, which makes the learned binary codes less effective. In this paper, we present an equivalent continuous formulation of the discrete hashing problem. Specifically, we show that the discrete hashing problem can be transformed into a continuous optimization problem without any relaxations, while the transformed continuous optimization problem has the same optimal solutions and the same optimal value as the original discrete hashing problem. After transformation, the continuous optimization methods can be applied. We devise the algorithms based on the idea of DC (difference of convex functions) programming to solve this problem. The proposed continuous hashing scheme can be easily applied to the existing hashing models, including both supervised and unsupervised hashing. We evaluate the proposed method on several benchmarks and the results show the superiority of the proposed method compared with the state-of-the-art hashing methods.

Index Terms—Continuous hashing, DC programming, large-scale image retrieval, learning to hash (LH).

I. INTRODUCTION

RECENTLY, hashing-based approximate nearest neighbors search has attracted substantial attention in various fields, including large-scale image retrieval, machine learning, and computer vision [1]–[11]. Hashing method encodes high-dimensional data points into compact binary codes, and meanwhile maps similar data points to adjacent binary hash codes to preserve the similarity in the original space. Using the binary codes to represent the massive data, nearest-neighbor search can be easily accomplished with a constant time complexity.

Hashing methods can be mainly divided into data-independent hashing [2], [15], [16] and data-dependent hashing [6], [10], [12], [31], [36]. One of the most popular data-independent hashing methods is locality sensitive hashing (LSH) [23], which generates the hash functions by random projections. Though LSH can map similar samples to similar codes with high probability, it usually requires long bits to achieve good precision. Such a drawback restricts its application.

In recent years, the learning-based data-dependent hashing methods have become more popular than the data-independent hashing methods, for the benefit that they can effectively and efficiently learn very compact binary codes of massive data. Different from the data-independent hashing, data-dependent hashing methods learn binary codes and hash functions from data, so they are also called learning to hash (LH) methods. Representative LH approaches include spectral hashing [7], iterative quantization (ITQ) [6], supervised discrete hashing (SDH) [12], structure sensitive hashing [14], and hashing with angular reconstructive embeddings [47]. More recently, Do et al. [20], Liong et al. [21], and Liu et al. [22] used deep networks to learn binary hash codes and achieved promising performance. He et al. [24] proposed a novel hashing scheme which can significantly save the training cost and meanwhile preserve the semantic similarity of the original data. Leng et al. [26], Wang et al. [27], and Liu et al. [28] proposed distributed hashing methods to deal with the large-scale image retrieval problems. The LH methods can be further divided into unsupervised hashing [5], [17], [23], [25], [29], [48]–[50] and supervised hashing [10], [12], [19], [30], [34], [37], [53]. The main difference between unsupervised hashing and supervised hashing is whether the label information is available for learning hash functions.

In general, the data-dependent hashing methods (including both unsupervised and supervised hashing) need to solve a discrete optimization problem to learn hash functions and hash codes, which is NP-hard. In the existing literature, most hashing methods choose to relax the problem by dropping the discrete constraints [7], [38]. Such a scheme brings much convenience and greatly simplifies the original problem. However, the relaxed problem can only obtain an approximate solution, which is proved to be less effective and of low quality, especially when the codes have long bits. In this situation, some discrete optimization methods [12], [17], [18] are
A. General Hashing Problem

Let \( X = [x_1, x_2, \ldots, x_n] \in \mathbb{R}^{d \times n} \) denote the data set of \( n \) samples. The aim of hashing is to encode each high-dimensional data point \( x \) into a compact binary code \( b \in \{-1, 1\}^c \). Generally, such a task is finished by solving the following binary code learning problem:

\[
\min_{B} \mathcal{L}(B) \quad \text{s.t.} \quad B \in \{-1, 1\}^{c \times n}
\]

where \( B = [b_1, b_2, \ldots, b_n] \) are the binary codes of the whole data and \( \mathcal{L}(B) \) is the objective function which is determined by the specific hashing model.

The above problem is a discrete optimization problem, which is NP-hard. Traditional hashing methods relax the problem by discarding the discrete constraints. As stated before, such a manner will bring large accumulated quantization error and make the binary codes less effective.

B. Continuous Formulation of Discrete Hashing

The relaxed method directly removes the discrete constraints, which makes the relaxed problem quite different from the original problem. In other words, the optimal solution of the relaxed problem is usually different from that of the original problem. Completely ignoring the discrete constraints is inadvisable. In this section, we present an equivalent continuous formulation of the discrete binary codes learning problem. We show that the discrete binary codes learning problem can be transformed into a continuous optimization problem without any relaxations, and the transformed continuous optimization problem has the same optimal solutions and the same optimal value as the original discrete optimization problem.
problem to obtain high-quality solutions. As we can see, in the literature, there are more efficient methods that can be used for continuous optimization, while the methods for discrete optimization are relatively rare and the performance is usually not satisfactory.

In this section, we consider a relatively simple case: the hashing problem without bits uncorrelation and balance constraints, that is, problem (1). To reformulate problem (1) in a continuous optimization form, we need the following lemma.

Lemma 1 [41]: Suppose that $f$ is Lipschitz continuous on $X$ with constant $L$. Let $\phi$ be a non-negative function defined on $X$ and $S := \{x \in X : \phi(x) = 0\}$. If $d(x, S) \leq \phi(x)$ for all $x \in X$, then for all $\gamma \geq L$, the two problems

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

are equivalent, that is, the two problems have the same optimal solutions and the same optimal value. Here $d(x, S) := \inf_{z \in S} ||x - z||$ denotes the distance from a point $x$ to the set $S$.

Compared with the hashing problem having the bits uncorrelation and balance constraints, problem (1) is relatively simple, and the process of transforming problem (1) into a continuous optimization problem is also relatively easy. In the following, we present the specific transformation method.

Let

$$B_1 = \{B \in \mathbb{R}^{c \times n} : B \in [-1, 1]^{c \times n}\}$$

and

$$B_2 = \{B \in \mathbb{R}^{c \times n} : B \in [-1, 1]^{c \times n}\}.$$

For all $x \in B_1$, we have that

$$d^2(x, B_2) = \min \left\{ \sum_{k=1}^{c} \sum_{i=1}^{n} (x_{k,i} - y_{k,i})^2 : y \in B_2 \right\} = \sum_{k=1}^{c} \sum_{i=1}^{n} (1 - |x_{k,i}|)^2. \tag{3}$$

Thus,

$$d(B, B_2) = \sqrt{\sum_{k=1}^{c} \sum_{i=1}^{n} (1 - |B_{k,i}|)^2}$$

$$\leq \sum_{k=1}^{c} \sum_{i=1}^{n} (1 - |B_{k,i}|)$$

$$\leq \sum_{k=1}^{c} \sum_{i=1}^{n} (1 - B_{k,i}^2)$$

$$= cn - \text{tr}(BB^T) = \phi_1(B). \tag{4}$$

for all $B \in B_1$. It is easy to verify that $\phi_1(B) \geq 0$ and $B \in B_2$ iff $\phi_1(B) = 0$, for all $B \in B_1$.

According to Lemma 1, we have that problem (1) is equivalent to the following continuous optimization problem:

$$\min_{B \in B_1} \mathcal{L}(B) + \gamma \phi_1(B) \tag{5}$$

for all $\gamma \geq L$, where $L$ is the Lipschitz constant of $\mathcal{L}(B)$ on $B_1$.

This scheme can be applied to most existing hashing methods, including both supervised and unsupervised hashing. For example, Shen et al. [12] proposed a supervised hashing model which learns binary codes by optimizing

$$\min_{B, \nu} \sum_{i=1}^{n} ||y_i - W^T b_i||^2 + \nu ||W||^2$$

s.t. $b_i \in [-1, 1]^c$ \tag{6}

where $y_i \in [0, 1]^c$ is the label of data $x_i$, $W \in \mathbb{R}^{c \times r}$ is the classification matrix, and $\nu$ is the regularization parameter. The above problem (6) can be rewritten as the matrix form

$$\min_{B, \nu} ||Y - W^T B||^2 + \nu ||W||^2$$

s.t. $B \in [-1, 1]^{c \times n}$. \tag{7}

where $Y = [y_1, \ldots, y_n] \in \mathbb{R}^{r \times n}$ is the label matrix. Using transformation approach shown above, the continuous equivalent of (7) can be represented as

$$\min_{B, \nu} ||Y - W^T B||^2 + \nu ||W||^2 + \gamma \phi_1(B)$$

s.t. $B \in [-1, 1]^{c \times n}$. \tag{8}

The unsupervised hashing problem can also be equivalently formulated as a continuous optimization problem, for example, the unsupervised graph hashing [7], which learns binary codes by solving the following problem:

$$\min_{B} \mathcal{L}(B) = \text{tr}(B(D - M)B^T)$$

s.t. $B \in [-1, 1]^{c \times n}$. \tag{9}

where $M_{n \times n}$ is the affinity matrix with $M_{i,j} = \exp(-||x_i - x_j||^2/\epsilon)$ and $D$ is a diagonal $n \times n$ matrix with $D_{i,i} = \sum_j M_{i,j}$. Here, the parameter $\epsilon > 0$ defines the distance in $\mathbb{R}^d$ which corresponds to similar items. Note that the time complexity to compute $M$ is $O(n^2)$, which is unacceptable in large-scale applications. In the real application, we will construct an anchor-based affinity matrix [5] $M = ZZ^T$, where $Z \in \mathbb{R}^{n \times m}$ is a truncated similarity matrix. The time complexity to construct $M$ by $Z$ is only $O(nm)$ with $m$ anchors $(m \ll n)$.

Applying the proposed continuous transformation method, we give the continuous formulation of problem (9)

$$\min_{B \in B_1} \text{tr} \left( B(D - M)B^T \right) + \gamma \phi_1(B). \tag{10}$$

Remark 1: The term $\phi_1(B)$ can be seen as the penalty which is used to eliminate the quantization error and force the optimal solution of the transformed problem to be in the original domain $B_2$. The condition $\gamma \geq L$ is only a sufficient condition but not necessary. Usually, a smaller $\gamma$ is enough to guarantee the equivalence of problem (5) and problem (1), especially when $B$ is large scale and correspondingly the Lipschitz constant $L$ is usually very large. Though a large $\gamma$ can always ensure that the optimal solution of (5) is the same as that of (1), since $\phi_1(B)$ is concave, large $\gamma$ will also add the difficulty of optimizing (5). In the experiment section, we will show that a moderate $\gamma$ is enough to guarantee the quantization error to be zero.
C. Continuous Formulation of Hashing With Bits Uncorrelation and Balance Constraints

To make the binary codes efficient, most hashing methods will add the bits uncorrelation and balance constraints. With these constraints, the hashing problem becomes

\[
\min_B \mathcal{L}(B) \\
\text{s.t. } B \in \{-1, 1\}^{c \times n} \\
B1 = 0 \\
BB^T = nl. \tag{11}
\]

From the last section, we see that, by introducing \(\varphi_1(B)\), the discrete hashing problem (1) can be transformed into a continuous optimization problem. However, \(\varphi_1(B)\) can only transform the discrete domain \(\mathcal{B}_2\) into the continuous domain \(\mathcal{B}_1\), but it cannot handle the additional bits uncorrelation and balance constraints. In this section, we will show that by constructing suitable \(\varphi(B)\), the bits uncorrelation and balance constraints can also be eliminated, and the above problem (11) can be transformed into an equivalent continuous unconstrained optimization problem. Meanwhile, the transformed problem still has the same optimal solutions and the same optimal value as the original problem (11). Since the two constraints make the problem more complex, it is difficult to directly construct \(\varphi(B)\) like the last section. In the following, we give the method that constructs \(\varphi(B)\) step by step, and one can see that due to the existence of the uncorrelation and balance constraints, \(\varphi(B)\) is complex and quite different from \(\varphi_1(B)\).

Let

\[
\mathcal{V} = \{B \in \mathbb{R}^{c \times n} : B \in \{-1, 1\}^{c \times n}, B1 = 0\}
\]

\[
\mathcal{W} = \{B \in \mathbb{R}^{c \times n} : B \in \{-1, 1\}^{c \times n}, B1 = 0, BB^T = nl\}.
\]

From Lemma 1, we see that the key point to construct \(\varphi(B)\) is to compute the upper bound of \(d(B, \mathcal{W})\) for all \(B \in \mathcal{B}_1\). In the following, we give the upper bound of \(d(B, \mathcal{W})\), \(\forall B \in \mathcal{B}_1\) step by step.

**Lemma 2:** For all \(B \in \mathcal{B}_2\)

\[
d(B, \mathcal{V}) \leq ||B1||.
\]

**Proof:** Let \(\mathcal{X} := \{x \in \mathbb{R}^n : x \in \{-1, 1\}^n\}\) and let \(\mathcal{Y} := \{y \in \mathbb{R}^n : y \in \{-1, 1\}^n, y^T1 = 0\}\). Let \(a^+\) denote the number of the components of \(x\) equaling to 1, and \(a^-\) be the number of the components of \(x\) equaling to -1. Without loss of generality, we assume \(a^+ \geq a^-\). Then one has that for all \(x \in \mathcal{X}\)

\[
d^2(x, y) = 4\left(a^+ \frac{n}{2} - a^+ + a^- \right) = 2(a^+ - a^-)^2 \leq (a^+ - a^-)^2 = (x'1)^2.
\]

The inequality holds because \(a^+ - a^- = 0\) or \(a^- - a^+ \geq 2\). Therefore, for all \(B \in \mathcal{B}_2\)

\[
d(B, \mathcal{V}) \leq ||B1||. \tag{12}
\]

Lemma 2 is proved.

Let \(\varphi_2(B) = ||B1||.\) It is easy to see that \(\varphi_2(B) = 0\) iff \(B \in \mathcal{V}\), for all \(B \in \mathcal{B}_2\), and \(\varphi_2(B)\) is Lipschitz continuous on \(\mathcal{B}_1\).

**Lemma 3:** Let \(\tau_1\) be the Lipschitz constant of \(\varphi_2(B)\) on \(\mathcal{B}_1\), then one has that

\[
d(B, \mathcal{V}) \leq (1 + \tau_1)\varphi_1(B) + \varphi_2(B), \forall B \in \mathcal{B}_1.
\]

**Proof:** Since \(\varphi_2(B)\) is Lipschitz continuous on \(\mathcal{B}_1\) with constant \(\tau_1\), we have that

\[
\varphi_2(B) \leq \varphi_2(B_2) + \tau_1 d(B_1, B_2), \forall B_1, B_2 \in \mathcal{B}_1.
\]

Let \(B \in \mathcal{B}_1\) and let \(B' \in \mathcal{B}_2\) be the projection of \(B\) on \(\mathcal{B}_2\). Then we have

\[
d(B, \mathcal{V}) \leq d(B, \mathcal{B}_2') + d(B', \mathcal{V}) \leq d(B, \mathcal{B}_2) + \varphi_2(B') \leq (1 + \tau_1)d(B_1, B_2') + \varphi_2(B) = (1 + \tau_1)d(B_1, B_2) + \varphi_2(B) \leq (1 + \tau_1)\varphi_1(B) + \varphi_2(B).
\]

The lemma is proved. □

Let \(\xi(B) = (1 + \tau_1)\varphi_1(B) + \varphi_2(B),\) then one has that for all \(B \in \mathcal{B}_1\), \(d(B, \mathcal{V}) \leq \xi(B)\) and \(\xi(B) = 0\) iff \(B \in \mathcal{V}\). Next, we give an upper bound of \(d(B, \mathcal{W})\), \(\forall B \in \mathcal{V}\).

**Lemma 4:** For all \(B \in \mathcal{V}\), one has that

\[
d(B, \mathcal{W}) \leq \zeta(||BB^T|| - n\sqrt{c})
\]

where \(\zeta = n^{3/2}c^{3/2}\).

**Proof:** It is easy to see that

\[
d(B, \mathcal{W}) = \min ||B - H||_F : H \in \mathcal{W} \leq 2\sqrt{cn}
\]

for all \(B \in \mathcal{V}\). Let \(x_1\) and \(x_2\) be two vectors from \(D := \{x \in \mathbb{R}^n : x \in \{-1, 1\}^n, x^T1 = 0\}\). If \(x_1^T x_2 \neq 0\), then it is easy to see that \(|x_1^T x_2| \geq 2\). Therefore, for \(B \in \mathcal{V}\), if \(B \notin \mathcal{W}\), we have that \(BB^T \neq nl\) and

\[
||BB^T||_F^2 - n^2c \geq 4.
\]

Thus, for all \(B \in \mathcal{V}\), we have that

\[
d(B, \mathcal{W}) \leq \sqrt{cn}\left(||BB^T||_F^2 - n^2c\right)/2. \tag{13}
\]

Note that

\[
||BB^T||_F^2 - n^2c = (||BB^T|| + n\sqrt{c}) \ast (||BB^T|| - n\sqrt{c})
\]

and

\[
||BB^T|| + n\sqrt{c} \leq nc + n\sqrt{c} \leq 2nc
\]

then

\[
||BB^T||_F^2 - n^2c \leq 2nc \ast (||BB^T|| - n\sqrt{c}).
\]

Therefore,

\[
d(B, \mathcal{W}) \leq n^{3/2}c^{3/2}(||BB^T||_F - n\sqrt{c}).
\]

Lemma 4 is proved. □

Let \(\varphi_3(B) = ||BB^T|| - n\sqrt{c}\). It is easy to verify that \(\varphi_3(B) = 0\) iff \(B \in \mathcal{W}\), for \(B \in \mathcal{V}\). □
WANGLet \(\tau_2\) be the Lipschitz constant of \(\zeta \varphi_3(B)\) on \(B_1\), then for all \(B \in B_1\), one has that
\[
d(B, \mathcal{V}) \leq (1 + \tau_2)\zeta(B) + \zeta \varphi_3(B) = (1 + \tau_2)[(1 + \tau_1)\varphi_1(B) + \varphi_2(B)] + \zeta \varphi_3(B).
\]
The proof of Lemma 5 is the same to the proof of Lemma 3, so we omit it.
Let \(\varphi(B) = (1 + \tau_2)[(1 + \tau_1)\varphi_1(B) + \varphi_2(B)] + \zeta \varphi_3(B)\). One has that
\[
d(B, \mathcal{V}) \leq \varphi(B)
\]
and \(\varphi(B) = 0\) iff \(B \in \mathcal{V}\), for all \(B \in B_1\). According to Lemma 1, we derive that problem (11) is equivalent to the following unconstrained continuous optimization problem:
\[
\min_B \mathcal{L}(B) + \gamma \varphi(B) = \mathcal{L}(B) + \gamma (1 + \tau_2)[(1 + \tau_1)\varphi_1(B) + \varphi_2(B)] + \zeta \varphi_3(B)
\]
s.t. \(B \in [-1, 1]^{c \times n}\). (14)
For convenience, we can represent problem (14) by
\[
\min_B \mathcal{L}(B) + \eta_1 \varphi_1(B) + \eta_2 \varphi_2(B) + \eta_3 \varphi_3(B)
\]
s.t. \(B \in [-1, 1]^{c \times n}\). (15)
where \(\eta_1, \eta_2\) and \(\eta_3\) are the parameters. Compared with (5), the bits uncorrelation and balance constraints are eliminated by introducing \(\varphi_2(B)\) and \(\varphi_3(B)\).

Also, we can apply the above scheme to existing hashing models with bits uncorrelation and balance constraints. First, we consider the supervised case. We consider that the objective function \(\mathcal{L}(B)\) in (11) has the same formulation as the objective function in (7), that is, \(\mathcal{L}(B) = ||Y - W^TB||^2 + \nu||W||^2\). Then the continuous formulation of the supervised hashing with bits uncorrelation and balance constraints can be represented as
\[
\min_{B \in B_1} \|Y - W^TB\|^2 + \nu\|W\|^2 + \eta_1 \varphi_1(B) + \eta_2 \varphi_2(B) + \eta_3 \varphi_3(B).
\]
Similarly, in the unsupervised case \(\mathcal{L}(B) = \text{tr}(B(D - M)B^T)\), the continuous formulation can be represented as
\[
\min_{B \in B_1} \text{tr}(B(D - M)B^T) + \eta_1 \varphi_1(B) + \eta_2 \varphi_2(B) + \eta_3 \varphi_3(B).
\]
(16)
(17)

Remark 2: In the hashing problem (11), the bits uncorrelation and balance constraints are introduced to make the code length short (more efficient). From (15), we see that the terms \(\eta_2 \varphi_2(B)\) and \(\eta_3 \varphi_3(B)\) force the variable to satisfy the bits uncorrelation and balance constraints. The larger the parameters \(\eta_2, \eta_3\) are, the more the two constraints are satisfied. Different from the traditional constrained optimization problems in which the constraints should be completely satisfied, in the hashing problems, the bits uncorrelation and balance constraints are added to make the binary codes efficient, which are not treated as hard constraints that cannot be violated. In the hashing problem, we preferentially minimize the objective function to make the binary codes effective. Shen et al. [12] showed that excessively emphasizing the two constraints will reduce the effectiveness of the solution. In Section IV, we will also show that large \(\eta_2, \eta_3\) are not conducive to getting good binary codes. In real applications, in order to achieve a tradeoff between effectiveness and efficiency, we can tune the parameters \(\eta_2, \eta_3\) with cross-validation techniques.

III. ALGORITHMS FOR SOLVING EQUIVALENT CONTINUOUS HASHING PROBLEMS
In the above section, we have transformed the discrete hashing problems into continuous optimization problems, then many continuous optimization methods can be used to obtain the binary codes and hash functions. In this section, we propose some algorithms based on the idea of DC programming to solve the above continuous optimization problems. First, we introduce some preliminaries of DC programming.

A. Brief Introduction of DC Programming
DC programming [42] is very suitable for solving smooth or nonsmooth nonconvex continuous optimization problem. The main idea of DC programming is to decompose the objective function \(f(x)\) as
\[
\min f(x) = g(x) - h(x), x \in \mathbb{R}^d
\]
where both \(g\) and \(h\) are convex functions. For a constrained optimization problem
\[
\min f(x) = g(x) - h(x), x \in C
\]
where \(C \subseteq \mathbb{R}^d\) is a convex set, by introducing the indicator function, the above problem can be formulated as the standard DC form
\[
\min g(x) + h(x), x \in \mathbb{R}^d
\]
where
\[
I_C(x) = \begin{cases} 
0, & x \in C \\
\infty, & \text{else}.
\end{cases}
\]

DC programming solves the above problems by DC algorithm (DCA), which quite often gives global solutions. DCA is proved to be more robust and more efficient than other related standard methods, especially in the large-scale setting [42]-[45]. The generic DCA scheme for solving (18) can be formulated as Algorithm 1. Here, \(\partial h(x)\) denotes the subgradient of \(h(x)\) defined as
\[
\partial h(x) = \{y \in \mathbb{R}^d : h(w) - h(x) \geq \langle w - x, y \rangle, \forall w \in \text{dom } h\}.
\]
If \(h(x)\) is differentiable, then \(\partial h(x)\) is just the gradient of \(h(x)\).

B. Algorithm for Solving Problem (8)
Like [12], we optimize problem (8) by alternatively updating the variables \(W\) and \(B\). With fixed \(B\), the variable \(W\) is updated by
\[
W = (BB^T + \nu I)^{-1}BY^T.
\]
(19)
Algorithm 1 DCA

Initialization

Let \( x^0 \in C \) be a starting point, and set \( k = 0 \).

Repeat

\[ \begin{align*}
& \text{Choose } y^k \in \partial h(x^k); \\
& \text{Choose } x^{k+1} \in \arg\min \{ g(x) - [h(x^k) + \langle x - x^k, y^k \rangle]; x \in C \}; \\
& \quad k = k + 1; \\
\end{align*} \]

Until Convergence condition holds.

Algorithm 2 CSH

Initialization

Input training data \( \{X, Y\} \); classification matrix \( W \); code length \( c \); parameter \( \gamma \). Initialize the starting point \( B^0 \) by the sign of random Gaussian matrix. Set \( k = 0 \).

Repeat

\[ \begin{align*}
& \text{Compute } A^k = 2(1 + \gamma)B^k; \\
& \text{Update } B \text{ by } B^{k+1} = \mathcal{P}_{B_1}(2WW^T + 2I)^{-1}(2WY + A^k); \\
& \quad k = k + 1; \\
\end{align*} \]

Until converge or reach maximum iterations.

Output \( B = \text{sign}(B) \);

With fixed \( W \), the variable \( B \) is updated by optimizing
\[
\min_B ||Y - W^TB||^2 + \gamma \varphi_1(B) \\
\text{s.t. } B \in [-1, 1]^{n \times n}. \tag{20}
\]

Note that the objective function of (20) is nonconvex with respect to \( B \). We aim to solve problem (20) by DC programming. According to the last section, the key step to apply DC programming is to decompose the objective function of problem (20) into the form of the difference of convex functions like (18). We rewrite problem (20) as
\[
\begin{align*}
\min_B & \quad G(B) - H(B) \\
\text{s.t.} & \quad B \in [-1, 1]^{n \times n} \tag{21}
\end{align*}
\]

where \( G(B) = ||Y - W^TB||^2 + \text{tr}(BB^T) \), and \( H(B) = \text{tr}(BB^T) + \gamma(\text{tr}(BB^T) - cn) \). According to the generic DCA scheme shown in Algorithm 1, to solve problem (21), at each iteration \( k \), we need to compute
\[
A^k \in \partial H(B^k) = 2(1 + \gamma)B^k. \tag{22}
\]

Then, we compute \( B^{k+1} \) by solving the convex programming
\[
\min_{B \in B_1} ||Y - W^TB||^2 + \text{tr}(BB^T) - \text{tr}(B^T A^k) \tag{23}
\]

which has the analytical solution
\[
B^{k+1} = \mathcal{P}_{B_1}\left[(2WW^T + 2I)^{-1}(2WY + A^k)\right]. \tag{24}
\]

We summarize the procedure of updating \( B \) in Algorithm 2. In practice, we only need to alternatively update \( W \) and \( B \) 2 ~ 5 times.

After the binary codes are obtained by the algorithm, we then use them to generate a hash function so that a novel query \( x \in \mathbb{R}^d \) can be efficiently encoded into a binary code.

We adopt the widely used nonlinear hash function having the form [12], [51]
\[
F(x) = \text{sign}(P^T \phi(x)) \tag{25}
\]

where \( \phi(x) = [\exp(-\|x - a_1\|^2/\sigma), \ldots, \exp(-\|x - a_m\|^2/\sigma)]^T \) is an \( m \)-dimensional column vector obtained using the RBF kernel mapping and \( P \in \mathbb{R}^{m \times c} \) is a projection matrix that projects \( \phi(x) \) onto the low-dimensional space. The vectors \( a_1, \ldots, a_m \) are \( m \) anchor points randomly selected from the training samples and \( \sigma \) is the kernel width. When the binary codes \( B \) are obtained, the hash function is learned by minimizing
\[
\min_P ||B - P^T \phi(X)||^2. \tag{26}
\]

It is easy to see that the optimal solution is
\[
P = (\phi(X)\phi(X)^T)^{-1}\phi(X)B^T. \tag{26}
\]

Hash functions having the similar formulations as (25) are widely used in the literature, such as SDH [12], SGH [48], and KSH [10]. Such hash functions are demonstrated to have better performance compared with linear hash functions, especially when the data is linearly inseparable [10].

C. Algorithm for Solving Problem (10)

To apply the DC programming, we also rewrite problem (10) as
\[
\begin{align*} 
\min_B & \quad G(B) - H(B) \\
\text{s.t.} & \quad B \in [-1, 1]^{n \times n} \tag{27}
\end{align*}
\]

where
\[
G(B) = \text{tr}(B(D - M)B^T) \tag{30}
\]

and
\[
H(B) = \gamma(\text{tr}(BB^T) - cn). \tag{31}
\]

Here, we adopt the anchor-based affinity matrix \( M = ZZ^T \) with the truncated similarity matrix \( Z \) as stated before. To optimize (27), according to DCA, at each iteration \( k \), we compute
\[
A^k \in \partial H(B^k) = 2\gamma B^k. \tag{28}
\]

Then, we compute \( B^{k+1} \) by solving the convex programming
\[
\min_{B \in B_1} \text{tr}(B(D - M)B^T) - \text{tr}\left(B^T A^k\right). \tag{29}
\]

We see that the above problem has the analytical solution
\[
B^{k+1} = \mathcal{P}_{B_1}\left[A^k * (D - M)^{-1}/2\right]. \tag{30}
\]

However, the time complexity for computing \( (D - M)^{-1} \) is \( O(n^3) \), which is unacceptable in large-scale application. Instead, we use the projected gradient descent method to solve the subproblem (29), avoiding unacceptable computational complexity. The gradient of \( J(B) \) is
\[
\nabla J(B) = 2B(D - M) - A^k. \tag{31}
\]
Algorithm 3 CUH

**Initialization**
Input training data $X$, code length $c$, parameter $\gamma$. Construct the matrices $M$ and $D$ using the training data and anchor points. Initialize the starting point $B^0$ by the sign of random Gaussian matrix. Set $k = 0$.

**Repeat**
Compute $A^k = 2\gamma B^k$;
Compute the gradient $\nabla J(B^k) = 2B^k(D - M) - A^k$;
Update $B^{k+1} = \mathcal{P}_{B^1}[B^k - \lambda \nabla J(B^k)]$;
$k = k + 1$;

**Until** converge or reach maximum iterations.

**Output** $B = \text{sign}(B)$;

The optimization procedure is summarized in Algorithm 3.

Here, $\lambda$ is a constant stepsize. At each iteration in Algorithm 3, we only need to update variable $B$ once using the projected gradient descent.

After the binary codes are obtained, we can compute the projection matrix $P$ using (26) to form the hash function $F(x)$ by (25).

**D. Algorithm for Solving Problem (15)**

As stated before, the bits uncorrelation and balance constraints are widely used in the previous hashing methods. In this section, we aim to solve problem (15) by DC programming. The term $\mathcal{L}(B)$ in (15) depends on the specific hashing model. For the supervised hashing (16), $\mathcal{L}(B) = ||Y - \hat{W}^T B||^2$, and for the unsupervised graph hashing (17), $\mathcal{L}(B) = \text{tr}(B(D - M)B^T)$. We know that the key point to apply DC programming is to decompose the objective function of (15) into the form of the difference of convex functions.

Note that $\varphi_1(B)$ is concave, and $\varphi_2(B)$ and $\varphi_3(B)$ are convex, so we only need to decompose $\mathcal{L}(B)$ by

$$\mathcal{L}(B) = \mathcal{L}_1(B) - \mathcal{L}_2(B)$$

with convex $\mathcal{L}_1(B)$ and $\mathcal{L}_2(B)$. In the literature, most existing hashing models have convex objective functions, including the above-mentioned two hashing models. In such cases, we can directly set $\mathcal{L}_1(B) = \mathcal{L}(B)$ and $\mathcal{L}_2(B) = 0$.

Now, we rewrite problem (15) as

$$\min_{B \in B_1} G(B) - H(B)$$

where

$$G(B) = \mathcal{L}_1(B) + \eta_2 \varphi_2(B) + \eta_3 \varphi_3(B)$$

$$= \mathcal{L}_1(B) + \eta_2 ||B1|| + \eta_3 (||BB^T|| - \sqrt{c})$$

and

$$H(B) = \mathcal{L}_2(B) - \eta_1 \varphi_1(B)$$

$$= \mathcal{L}_2(B) + \eta_1 (\text{tr}(BB^T) - c\eta).$$

Obviously, both $G(B)$ and $H(B)$ are convex. Next, we devise the algorithm based on DCA to solve problem (32). At each iteration $k$, we compute

$$A^k \in \partial H(B^k) = \nabla \mathcal{L}_2(B) + 2\eta_1 B^k.$$  

Algorithm 4 CCH

**Initialization**
Input training data $X$, code length $c$, parameters $\eta_1, \eta_2, \eta_3$. Initialize the starting point $B^0$ by the sign of random Gaussian matrix. Set $k = 0$.

**Repeat**
Compute $A^k = \nabla \mathcal{L}_2(B_k) + 2\eta_1 B^k$;
Compute the gradient $\nabla J(B^k)$ by (35) with $B = B_k$;
Update $B^{k+1} = \mathcal{P}_{B_1}[B^k - \lambda \nabla J(B^k)]$;
$k = k + 1$;

**Until** converge or reach maximum iterations.

**Output** $B = \text{sign}(B)$;

Then, we obtain $B^{k+1}$ by solving the convex programming

$$\min_{B \in B_1} J(B) = G(B) - \text{tr}(B^T A^k).$$  

It is not easy to find an analytical solution of problem (34). Similar to problem (29), we can solve this problem using projected gradient descent method. The gradient of $J(B)$ is computed as

$$\nabla J(B) = \nabla \mathcal{L}_1(B) + \eta_2 \nabla \varphi_2(B) + \eta_3 \nabla \varphi_3(B) - A^k$$

where

$$\nabla \varphi_2(B) = B \ast 11^T/||B1||$$

$$\nabla \varphi_3(B) = 2BB^T/||BB||$$

and $\mathcal{L}_1(B)$ is determined by the specific hashing model. The optimization procedure is summarized in Algorithm 4.

After obtaining the binary codes, we compute the projection matrix $P$ by (26) and form the hash function $F(x)$ by (25).

Remark 3: The continuous constrained hashing (CCH) algorithm is used for solving the discrete hashing problem with bits uncorrelation and balance constraints, that is, problem (11). Problem (11) is a general model and CCH is a general algorithm framework. The CCH algorithm can be used for solving both the supervised and unsupervised hashing problems. For clarity, in this paper, we respectively, use CCH-S and CCH-U to represent the CCH algorithm for solving the supervised hashing problem (16) and the unsupervised hashing problem (17).

IV. EXPERIMENTS

In this section, we use numerical experiments to show the performance of the proposed algorithms. We evaluate our method on three benchmark datasets: 1) CIFAR-10 [12]; 2) MNIST [34]; and 3) NUS-WIDE [5]. For each dataset, we randomly select 1000 data points from the training data as the anchor points. The baselines for comparison include the supervised hashing: SDH [12], FSDH [33], COSDISH [30], CCA-ITQ [6], KSH [10], and LFH [39]; and the unsupervised hashing: IMH [32], AGH [5] SGH [48], OCH [36], and PCA-ITQ [6]. For each method, we use the same parameters setting provided by the corresponding papers. We use CCH-U and CCH-S to, respectively, denote the CCH algorithm in the unsupervised and supervised cases as stated before. In the following
examples, we empirically set $\lambda = 5$, $\gamma = 1$, $\eta_1 = 2$, $\eta_2 = 10$, and $\eta_3 = 10$, and we set the maximum iteration number $t = 20$.

All the results below are averaged over 50 independent runs.

### A. CIFAR-10: Test on Tiny Natural Images

In this example, we choose the dataset CIFAR-10 to test the performance of the proposed method. CIFAR-10 consists of 60K labeled images evenly divided in ten classes. Each image is represented by a 512-D GIST descriptor extracted from a $32 \times 32$ color image. We randomly select 1000 points as queries and use the rest as the training set. Since each data point from CIFAR-10 has the label information, both supervised and unsupervised hashing algorithms can be applied. The ground truth neighbors of each query are determined based on the label agreement.

For evaluation, we report the results in terms of mean average precision (MAP) and mean precision of the top 500 retrieved neighbors based on Hamming ranking in Table I with the code length varying from 8 bits to 256 bits.

From Table I, we see that the proposed continuous supervised hashing (CSH), continuous unsupervised hashing (CUH), and CCH algorithms outperform the other hashing methods in most cases. We also see that CCH-S and CCH-U, respectively, achieve better results than CSH and CUH, which implies that the bits uncorrelation and balance constraints can improve the quality of the binary codes.

We further report the training time consumption of the hashing methods in Table II. The results show that our algorithms are time efficient. The training time of the proposed methods is much less than that of KSH, SDH, SGH, and COSDISH, and is close to that of IMH, AGH, PCA-ITQ, OCH, CCA-ITQ, LFH, and FSDH.

Next, we investigate the sensitivity of the parameters $\eta_2$ and $\eta_3$. Fig. 1 shows the MAP results of CCH-U and CCH-S on CIFAR-10 with 64 bits with respect to different values of $\eta_2$ and $\eta_3$. We can see that the moderate $\eta_2$, $\eta_3$ can help to improve the retrieval accuracy, while the large $\eta_2$, $\eta_3$ will reduce the retrieval accuracy on the contrary, which demonstrates that excessively emphasizing the two constraints will reduce the quality of the solution.

The parameter $\gamma$ in CSH and CUH and the parameter $\eta_1$ in CCH-S and CCH-U mainly affect the quantization error. We plot the curves of the quantization error with respect to the parameters $\gamma$, $\eta_1$ with 64 bits in Fig. 2. From the figure, we see
that moderate $\gamma, \eta_1$ are enough to guarantee the quantization error to be very small, and when $\gamma = 1$ and $\eta_1 = 2$, the quantization error is very close to zero. The results shown in the figure are in line with the statement in Remark 1.

**B. MNIST: Evaluation Using the Class-Wise Splitting Protocol [52]**

Recently, a class-wise splitting protocol [52] was proposed for evaluating supervised hashing, and it is demonstrated that the class-wise splitting protocol can better capture desirable properties of supervised hashing schemes, compared with the traditional protocols. So, in this example, we use this protocol to further test the performance of the proposed method on supervised hashing. The dataset for test is MNIST, which consists of 70,000 images of handwritten digits from “0” to “9.” Each of the images is represented by a 784-D vector. The whole dataset is partitioned into a test set with 1000 samples and a training set with all remaining samples. The class-wise splitting protocol requires the hashing method to use disjoint set of classes for training and testing. Like [52], we separate the training set based on class splits into two disjoint sets, namely train70 and train30, where train70 contains 70% of the classes and train30 contains the remaining 30% of the classes. Also, we separate the testing set into test70 and test30 using the same method, and the classes in test70 are the same as those in train70. In this example, the set train70 is used for training. The set train30 is the database for retrieval and the set test30 is the query set. The set test70 is not used. We show the results in terms of MAP and the mean precision of the top 500 retrieved neighbors in Table III. The results show that the proposed methods outperform the other supervised hashing methods for comparison in most cases in terms of the class-wise splitting protocol.

**C. NUS-WIDE: Accuracy on Large-Scale Dataset**

In this example, we test the performance of the proposed method on the large-scale dataset NUS-WIDE, which consists of around 270,000 Web images associated with 81 ground truth concept labels and each image contains multiple semantic labels. Each image is represented by a 500-D bag-of-words feature. Any two images sharing at least one label are seen as true neighbors. We collect 21 most frequent labels and randomly select 100 images as queries for each label. We use the remaining images as the training set. Note that it is hard to separate the dataset based on class splits without overlap since each sample in NUS-WIDE belongs to several classes [54]. So, in this example, we use the same traditional evaluation protocol as Example 1. Since the dataset of this...
example is large-scale and it is very slow to compute MAP of the whole dataset, we only return top 1000 retrieved samples and calculate the MAP of the top 1000 returns. The results in terms of top-1000 MAP and mean precision of top 500 retrieved neighbors are, respectively, shown in Figs. 3 and 4. The results shown in the figures demonstrate the superiorities of the proposed methods.

V. CONCLUSION

In this paper, we gave an equivalent continuous formulation of the general discrete hashing problem. Specifically, we showed that the discrete hashing problem can be transformed into a continuous optimization problem without any relaxations, and the continuous optimization problem has the same optimal solutions and optimal value as the original problem. After transformation, we devised algorithms based on the idea of DC programming and DCA to solve the continuous optimization problem. We showed that such a scheme can be applied to both supervised and unsupervised hashing models. Experiments showed the superiority of the proposed method in terms of accuracy and time efficiency.

REFERENCES


**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 interests include machine learning 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, and quantitative finance.

**Hui-Liang Shen** received the B.Eng. and Ph.D. degrees in electronic engineering from Zhejiang University, Hangzhou, China, in 1999 and 2006, respectively.

He was a Research Associate and a Research Fellow with the 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.