PRELIMINARY PREPRINT VERSION: DO NOT CITE
The AAAI Digital Library will contain the published
version some time after the conference.

# DPCD: Discrete Principal Coordinate Descent for Binary Variable Problems

**Huan Xiong**[1,2]

[1]Harbin Institute of Technology, China
[2]MBZUAI, United Arab Emirates
huan.xiong.math@gmail.com

## Abstract

Binary optimization, a representative subclass of discrete optimization, plays an important role in mathematical optimization and has various applications in computer vision and machine learning. Generally speaking, binary optimization problems are NP-hard and difficult to solve due to the binary constraints, especially when the number of variables is very large. Existing methods often suffer from high computational costs or large accumulated quantization errors, or are only designed for specific tasks. In this paper, we propose an efficient algorithm, named **Discrete Principal Coordinate Descent (DPCD)**, to find effective approximate solutions for **general binary optimization problems**. The proposed algorithm iteratively solves optimization problems related to the linear approximation of loss functions, which leads to updating the binary variables that most impact the value of the loss functions at each step. Our method supports a wide range of empirical objective functions with/without restrictions on the numbers of 1s and $-1$s in the binary variables. Furthermore, the theoretical convergence of our algorithm is proven, and the explicit convergence rates are derived for objective functions with Lipschitz continuous gradients, which are commonly adopted in practice. Extensive experiments on several binary optimization tasks and large-scale datasets demonstrate the superiority of the proposed algorithm over several state-of-the-art methods in terms of both effectiveness and efficiency.

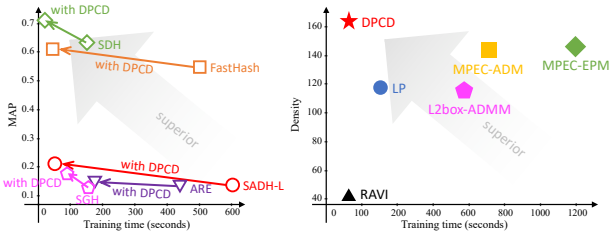
## 1 Introduction

Binary optimization problems are generally formulated as follows:

$$\min_{\mathbf{x}} \ f(\mathbf{x}), \qquad \text{s.t. } \mathbf{x} \in \{\pm 1\}^n. \tag{1}$$

Problem (1) appears naturally in several fields of computer vision and machine learning, including clustering [32], graph bisection [33, 38], image denoising [5], dense subgraph discovery [2, 4, 38, 41], multi-target tracking [30], and community discovery [9]. In many application scenarios, such as binary hashing [8, 19, 29, 31, 37], Problem (1) needs to be solved for millions of binary variables, which makes the size $2^n$ of the feasible set very large (far larger than the number of atoms in the universe). Usually, it is difficult to find the optimal solution. Therefore, providing a fast algorithm to approximately solve Problem (1) is important in practice.

Copyright © 2022, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.




(a) Hashing Retrieval on CIFAR10 with 96-bit codes

(b) Dense Subgraph Discovery on frwiki-2013 with k=1600

Figure 1: (a) Improvements of the proposed DPCD method for state-of-the-art hashing methods when applied to same specific loss functions and (b) comparison of DPCD with other general binary optimization methods on dense subgraph discovery problem.

Furthermore, additional constraints on the numbers of 1s and $-1$s in the binary variables $\mathbf{x}$ are adopted in many cases. For example, in binary hashing [28, 29] and graph bisection [33, 38], a balance condition is often required, which means that the numbers of 1s and $-1$s are equal to each other. On the other hand, dense subgraph discovery [2, 4, 38, 41] and information theoretic clustering [32] require that the numbers of 1s and $-1$s in $\mathbf{x}$ are some fixed integers.

To handle the previously mentioned constraints, in this paper, we focus on the following binary optimization problem:

$$\min_{\mathbf{x}} \ f(\mathbf{x}), \qquad \text{s.t. } \mathbf{x} \in \Omega_r, \tag{2}$$

where $\mathbf{x}$ is a binary vector of length $n$, $f(\cdot)$ is a differentiable objective function (which may be nonconvex), $r \geq -1$ is a given integer, $\mathbb{N}_0$ denotes the set of nonnegative integers, and the restriction $\Omega_r$ on $\mathbf{x}$ is defined as

$$\Omega_r = \begin{cases} \{\pm 1\}^n, & \text{if } r = -1; \\ \{\mathbf{x} \in \{\pm 1\}^n : \mathbf{1}^\top \mathbf{x} = 2r - n\}, & \text{if } r \in \mathbb{N}_0. \end{cases} \tag{3}$$

When $r = -1$, Problem (2) is a binary optimization problem without further constraints. When $r \in \mathbb{N}_0$, Problem (2) becomes an optimization problem with the restriction that there are exactly $r$ 1s in the binary vector $\mathbf{x}$. For instance, when $r = n/2$, the constraint $\mathbf{1}^\top \mathbf{x} = 2r - n = 0$ implies that the number of 1s is equal to the number of $-1$s in $\mathbf{x}$.

In general, Problem (2) is NP-hard due to the binary constraints [13]. Many algorithms, such as continuous relaxation, equivalent optimization, signed gradient optimization and

direct discrete optimization, have been proposed to approximate the solution (for details, please refer to Section 2). However, they usually suffer from high computational costs or large accumulated quantization errors, or are only designed for specific tasks.

To overcome these difficulties, in this paper, we propose a **fast** and **generalized** optimization algorithm, termed Discrete Principal Coordinate Descent (DPCD), to approximately solve Problem (2). The time complexity for the binary optimization problem is relatively high when directly applying signed gradient methods (updating all variables at each iteration based on the gradients). In contrast, the proposed DPCD updates the principal coordinates that have the highest impact on the value of the loss function at each iteration, which makes the updating procedure very fast. Different from other binary optimization algorithms in the literature, our DPCD method **supports a large family of empirical objective functions** with/without restrictions on the numbers of 1s and $-1$s in the binary variables. Furthermore, we prove the **theoretical convergence** of DPCD for loss functions with Lipschitz continuous gradients, which cover almost every loss function in practice. Explicit convergence rates are also derived. Extensive experiments on two binary optimization tasks (dense subgraph discovery and binary hashing) demonstrate the superiority of our method over state-of-the-art methods in terms of both efficiency and effectiveness. For example, Fig. 1 shows that DPCD can improve the performance (e.g., MAP, training time and values of loss function) of various binary optimization methods on the image retrieval and dense graph discovery tasks when applied to their specific loss functions.

## 2 Related Work

A very rich literature and a wide range of promising methods exist for binary optimization. We briefly review four classes of representative and related methods.

**Continuous relaxation methods.** An intuitive method for approximately solving Problem (1) is to relax the binary constraints to continuous variables, then threshold the continuous solutions to binary vectors. For instance, in the Linear Programming (LP) relaxation [10, 14], the binary constraint is substituted with the box constraint, i.e., $\mathbf{x} \in [-1, 1]^n$, which can be solved by continuous optimization methods, such as the interior-point method [22]. On the other hand, the Semi-Definite Programming (SDP) relaxation [33] replaces the binary constraint with some positive semi-definite matrix constraint. In spectral relaxation [17, 25], the binary constraint is relaxed to some $\ell_2$-ball, which is non-convex. One of advantages of such continuous relaxation methods is that the relaxed problems can be approximately and efficiently solved by existing continuous optimization solvers. However, the relaxation is usually too loose, and the thresholding often yields large quantization errors.

**Equivalent optimization methods.** Unlike relaxation methods, equivalent optimization methods replace the binary constraint with some equivalent forms, which are much easier to handle. For example, motivated by linear and spectral relaxations, Wu and Ghanem [36] replaced the binary constraint with the intersection of the box $[-1, 1]^n$ and the sphere $\{\mathbf{x} : \|\mathbf{x}\|_2^2 = n\}$, and then applied the Alternating Direction Method of Multipliers (ADMM) [6, 15, 34] to solve the optimization problem iteratively. Other methods in this direction include the MPEC-ADM and MPEC-EPM methods ([38, 40]), the $\ell_0$ norm reformulation [20, 39], the $\ell_2$ box non-separable reformulation [23], and the piecewise separable reformulation [43]. Usually, these equivalent optimization methods guarantee the convergence to some stationary and feasible points, but the convergence speed is often too slow, resulting in high computational costs for large-scale optimization problems.

**Signed gradient methods.** In the Signed Gradient Method (SGM) [19], a linear surrogate of the objective function $f(\mathbf{x})$ is given at each iteration. Then, the minimization (actually a maximization problem was studied in the original paper [19], we state an equivalent form here) of this surrogate function gives the updating rule for Problem (1) as: $\mathbf{x}^{k+1} = -\text{sgn}(\nabla f(\mathbf{x}^k))$. The sequence obtained by this updating rule is guaranteed to converge if the objective function is concave. However, even for a very simple non-concave function, SGM may generate a divergent sequence and never converge (please refer to Lemma 1). Furthermore, SGM cannot handle binary problems with restrictions on the number of 1s since this number may change during each iteration. A stochastic version of this method is Adaptive Discrete Minimization (ADM) [18], in which an adaptive ratio $\psi$ is selected at each iteration, then some random $\psi n$ entries of $\mathbf{x}$ are updated by $\mathbf{x}_i^{k+1} = -\text{sgn}(\nabla_i f(\mathbf{x}^k))$. Although ADM works well for certain loss functions, it fails when the value of the loss function depends largely on only a few variables, since the random selecting procedure may skip such important variables.

**Discrete optimization methods.** In the field of image hashing, many direct discrete optimization methods, such as DCC [28], SADH [29], ARE [11], SGH [12] and FastHash [16], have been proposed, which aim to directly optimize binary variables. It is well known that the Coordinate Descent (CD) [35] is widely used for solving optimization problems with smooth and convex constraints. Motivated by this method, several Discrete Cyclic Coordinate descent (DCC) methods (e.g., RDCM [21], FSDH [8], and SDH [28]) have been proposed to handle the binary constraint directly. The main idea is that, at each iteration, we consider a subproblem with most entries of the binary variables fixed, and minimize the loss function with respect to the remaining entries. Although such methods can work well for specific loss functions, they are generally difficult to extend to general binary optimization problems. Furthermore, they often suffer from expensive computational costs.

## 3 Discrete Principal Coordinate Descent

In this section, we present in detail the DPCD algorithm for solving Problem (2), which is a general binary optimization problem with/without restrictions on the numbers of 1s and $-1$s. We also provide a theoretical convergence analysis.

### 3.1 Notation and preliminaries

We first introduce several notations and preliminaries. A vector is represented by some lowercase bold character, while a matrix is represented by some uppercase bold character. Let $\mathbf{x}_i$ and $\mathbf{A}_{ij}$ denote the $i$-th and $(i,j)$-th entries of a vector $\mathbf{x}$ and a matrix $\mathbf{A}$, respectively. The transpose of a matrix $\mathbf{A}$ is represented by $\mathbf{A}^\top$. We use $\langle \cdot, \cdot \rangle$ to denote the Euclidean inner product. Let $\|\mathbf{A}\| = \sqrt{\sum_{ij} \mathbf{A}_{ij}^2}$ and $\|\mathbf{A}\|_1 = \sum_{ij} |\mathbf{A}_{ij}|$ be the Frobenius norm and 1-norm of a matrix $\mathbf{A}$, respectively. The gradient of a differentiable function $f(\mathbf{x})$ is denoted by $\nabla f(\mathbf{x}) = (\nabla_1 f(\mathbf{x}), \nabla_2 f(\mathbf{x}), \cdots, \nabla_n f(\mathbf{x}))$. Let $\mathrm{sgn}(\mathbf{x}) = (\mathrm{sgn}(\mathbf{x}_1), \mathrm{sgn}(\mathbf{x}_2), \cdots, \mathrm{sgn}(\mathbf{x}_n))$ denote the element-wise sign function where $\mathrm{sgn}(\mathbf{x}_i) = 1$ for $\mathbf{x}_i \geq 0$ and $-1$ otherwise. The Hamming distance between two binary vectors $\mathbf{y}$ and $\mathbf{z}$ of equal length is defined by $d_H(\mathbf{y}, \mathbf{z})$, which is the number of positions at which the corresponding entries are different. For a set $S$, let $\#S$ denote the number of elements in $S$.

Algorithm 1: Discrete Principal Coordinate Descent (DPCD)

**Input:** Loss function $f(\mathbf{x})$, code length $n$, the restriction $\Omega_r$ where $r = -1$ or $r \in \mathbb{N}_0$, parameters $\alpha_1$, $\alpha_2$, $\epsilon$.
**Output:** Binary codes $\mathbf{x}^*$.
Initialize $\mathbf{x}^*$ by the sign of some random vector according to $\Omega_r$; $\mathbf{x}^1 = \mathbf{x}^*$ and $k = 1$;
**while** *not converge or not reach maximum iterations* **do**
  Calculate $\nabla f(\mathbf{x}^k) = (\nabla_1 f(\mathbf{x}^k), \nabla_2 f(\mathbf{x}^k), \cdots, \nabla_n f(\mathbf{x}^k))$;
  Derive proper thresholds $L_1, L_2$ by Eq. (6) or Eq. (7);
  Build sets $S^{k+} = \{i : \nabla_i f(\mathbf{x}^k) > \alpha_1 L_1, \mathbf{x}_i^k = 1\}$ and $S^{k-} = \{i : \nabla_i f(\mathbf{x}^k) < -\alpha_2 L_2, \mathbf{x}_i^k = -1\}$;
  **if** *the restriction condition is* $\mathbf{x} \in \Omega_{-1}$ *(i.e.,* $r = -1$*)* **then**
    Update $\mathbf{x}_i^{k+1} = -\mathrm{sgn}(\nabla_i f(\mathbf{x}^k)) = -\mathbf{x}_i^k$ for $i \in S^{k+} \cup S^{k-}$;
  **else**
    Sort $\{|\nabla_i f(\mathbf{x}^k)| : i \in S^{k+}\}$ and $\{|\nabla_j f(\mathbf{x}^k)| : j \in S^{k-}\}$ in descending order as $|\nabla_{i_1} f(\mathbf{x}^k)| \geq |\nabla_{i_2} f(\mathbf{x}^k)| \geq |\nabla_{i_3} f(\mathbf{x}^k)| \geq \cdots$, and $|\nabla_{j_1} f(\mathbf{x}^k)| \geq |\nabla_{j_2} f(\mathbf{x}^k)| \geq |\nabla_{j_3} f(\mathbf{x}^k)| \geq \cdots$, respectively;
    Update $\mathbf{x}_{i_l}^{k+1} = -\mathbf{x}_{i_l}^k$ and $\mathbf{x}_{j_l}^{k+1} = -\mathbf{x}_{j_l}^k$ for $1 \leq l \leq \min\{\#S^{k+}, \#S^{k-}\}$;
  (Optional) Neighborhood search for $\mathbf{x}^{k+1}$;
  $k = k + 1$;
**Return** $\mathbf{x}^* = \mathbf{x}^{k+1}$.

### 3.2 Main algorithm

The proposed DPCD algorithm runs iteratively between the principal coordinate update and neighborhood search.

**Principal coordinate update.** The basic idea is that, at the $k$-th iteration, we change the sign of some adaptively chosen entries of the binary vector $\mathbf{x}^k$, such that the value of the loss function decreases steeply after each change. To achieve this goal, we focus on $L$-principal coordinates (see the following definition), which have major influences on the value change of the loss function.

**Definition 1.** *Let $f(\mathbf{x})$ be a differentiable function and $L > 0$ be a positive constant. A coordinate index $i$ is called an $L$-principal coordinate of $\mathbf{x} \in \{-1, 1\}^n$ if the product $\mathbf{x}_i \cdot \nabla_i f(\mathbf{x}) \geq L$.*

One motivation of our method is SGM [19]. At the $k$-th iteration of SGM, the linear surrogate of the objective function $f(\mathbf{x})$ is given as:

$$\hat{f}^k(\mathbf{x}) = f(\mathbf{x}^k) + \langle \nabla f(\mathbf{x}^k), \mathbf{x} - \mathbf{x}^k \rangle. \tag{4}$$

Then $\mathbf{x}^{k+1}$ is obtained by minimizing Eq. (4):

$$\mathbf{x}^{k+1} = \arg \min_{\mathbf{x} \in \{\pm 1\}^n} \hat{f}^k(\mathbf{x}) = -\mathrm{sgn}(\nabla f(\mathbf{x}^k)). \tag{5}$$

The sequence obtained by Eq. (5) is guaranteed to converge if $f(\mathbf{x})$ is concave. However, from Lemma 1 in Subsection 3.3 we know, for non-concave functions, SGM [19] may generate divergent sequences and never converge, since changing too many entries of $\mathbf{x}$ at a time may increase the value of the loss function. To overcome this difficulty, the proposed DPCD method only changes the signs of entries (entries with principal coordinates) whose absolute values of directional derivatives are large enough. This yields convergence for a wide class of loss functions (please refer to Lemma 1 and Theorem 1). Also, in the proposed algorithm, the constraint $\Omega_r$ is always satisfied after each iteration.

To be more specific, when $\mathbf{x}^k$ is given at the $k$-th iteration, we first calculate the gradient $\nabla f(\mathbf{x}^k) = (\nabla_1 f(\mathbf{x}^k), \nabla_2 f(\mathbf{x}^k), \cdots, \nabla_n f(\mathbf{x}^k))$ for the differentiable loss function $f(\mathbf{x})$. Next, we derive some proper thresholds $L_1, L_2$ based on the value of $\nabla f(\mathbf{x}^k)$. When $\nabla f$ is $L_0$-Lipschitz continuous on $[-1, 1]^n$, where $L_0$ is easy to calculate, we simply set

$$L_1 = L_2 = L_0 + \epsilon \tag{6}$$

for some sufficiently small positive constant $\epsilon > 0$, i.e., we consider $(L_0 + \epsilon)$-principal coordinates. For example, in Lemma 1, it is easy to see that $L_0 = 1$; then we take $L_1 = L_2 = 1 + \epsilon$ for some small $\epsilon > 0$ in the algorithm. When $L_0$ does not exist or is difficult to compute, we let $L_1$ and $L_2$ be averages of the absolute values of the positive and negative entries in the gradient $\nabla f(\mathbf{x}^k)$, respectively, i.e.,

$$\begin{cases} L_1 = \frac{1}{n_1} \sum_{\nabla_i f(\mathbf{x}^k) > 0} \nabla_i f(\mathbf{x}^k); \\ L_2 = -\frac{1}{n_2} \sum_{\nabla_i f(\mathbf{x}^k) < 0} \nabla_i f(\mathbf{x}^k), \end{cases} \tag{7}$$

where $n_1$ and $n_2$ are the numbers of positive and negative entries in $\nabla f(\mathbf{x}^k)$, respectively. With the given thresholds $L_1$ and $L_2$, we set $S^{k+} = \{1 \leq i \leq n : \nabla_i f(\mathbf{x}^k) > \alpha_1 L_1, \mathbf{x}_i^k = 1\}$ and $S^{k-} = \{1 \leq i \leq n : \nabla_i f(\mathbf{x}^k) < -\alpha_2 L_2, \mathbf{x}_i^k = -1\}$ to be the sets of $\alpha_1 L_1$-principal coordinates with positive partial derivatives and $\alpha_2 L_2$-principal coordinates with negative partial derivatives, respectively, where $\alpha_1$ and $\alpha_2$ are some parameters in $[0.1, 10]$ which are learned depending on the tasks and datasets. If the restriction condition is $\mathbf{x} \in \Omega_{-1}$, we update $\mathbf{x}_i^{k+1}$ by solving $\min_{\mathbf{x} \in \{\pm 1\}^n} \hat{f}^k(\mathbf{x})$ in Eq. (5) with respect to $i \in S^{k+} \cup S^{k-}$ (other entries of $\mathbf{x}$ are fixed) and derive:

$$\mathbf{x}_i^{k+1} = -\mathrm{sgn}(\nabla_i f(\mathbf{x}^k)) = -\mathbf{x}_i^k. \tag{8}$$

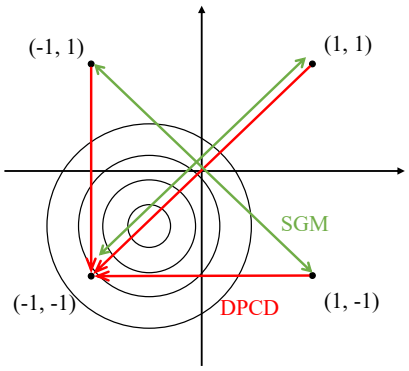



Figure 2: The optimization routes of DPCD and SGM on a 2-d example. The black circles are the contour lines of the objective function. Our method DPCD can converge to the optimum from any initialization points in only one step, while SGM always oscillates between two points. In fact, each $\beta_i$ can be in the open interval (-1,1), and the optimum is decided by signs of all $\beta_i$.

In other words, we change the sign of $\mathbf{x}_i^k$ for $\mathbf{x}_i^k = 1$ with $\alpha_1 L_1$-principal coordinates, and for $\mathbf{x}_i^k = -1$ with $\alpha_2 L_2$-principal coordinates. If the number of 1s in $\mathbf{x}$ is required to be fixed (i.e., the restriction condition is $\mathbf{x} \in \Omega_r$ for some $r \in \mathbb{N}$), we update Eq. (8) with respect to the $m$ largest absolute values in $\{|\nabla_i f(\mathbf{x}^k)| : i \in S^{k+}\}$ and $\{|\nabla_j f(\mathbf{x}^k)| : j \in S^{k-}\}$, respectively, where $m = \min\{\#S^{k+}, \#S^{k-}\}$ (such procedure guarantees that $\mathbf{x}^k \in \Omega_r$ implies $\mathbf{x}^{k+1} \in \Omega_r$). When the complexity of the gradient calculations is low, the updating is fast. *A complexity analysis of the proposed DPCD is given in Subsection 4.2, which shows that the algorithm complexity of DPCD for supervised discrete hashing is linear and thus the algorithm runs very fast for large-scale datasets.*

**Neighborhood search.** We add an optional heuristic neighborhood search after the principal coordinate update to avoid saddle points. In practice, we run one neighborhood search after $T$ principal coordinate updates, where $T$ is between 10 and 20. First, we define the concept of *$m$-neighbors* for a point $\mathbf{x} \in \Omega_r$, where $m \in \mathbb{N}$. When $r = -1$, the set of $m$-neighbors for $\mathbf{x} \in \Omega_{-1}$ is denoted by $N_{-1}(\mathbf{x}) := \{\mathbf{y} \in \{\pm 1\}^n : 0 < d_H(\mathbf{y}, \mathbf{x}) \leq m\}$, which is the set of points with a Hamming distance at most $m$ from $\mathbf{x}$. When $r \geq 0$, the set of $m$-neighbors for $\mathbf{x} \in \Omega_r$ is denoted by $N(\mathbf{x}) := \{\mathbf{y} \in \{\pm 1\}^n : 0 < d_H(\mathbf{y}, \mathbf{x}) \leq 2m,\ \sum_{i=1}^n \mathbf{y}_i = \sum_{i=1}^n \mathbf{x}_i\}$, which is the set of points obtained by interchanging at most $m$ pairs of 1 and $-1$ entries in $\mathbf{x}$. For instance, when $m = 1$, we have $N_{-1}((1, -1, 1)) = \{(-1, -1, 1), (1, 1, 1), (1, -1, -1)\}$ and $N((1, -1, 1, )) = \{(-1, 1, 1), (1, 1, -1)\}$.

In a neighborhood search for some $\mathbf{x} \in \Omega_{-1}$ (or $\Omega_r$ with $r \in \mathbb{N}_0$), the aim is to find some $\mathbf{y} \in N_{-1}(\mathbf{x}) \cup \{\mathbf{x}\}$ (or $N(\mathbf{x}) \cup \{\mathbf{x}\}$) with the minimal function value $f(\mathbf{y}) - f(\mathbf{x})$ (or equivalently, $f(\mathbf{y})$). In practice, we sample from $N_{-1}(\mathbf{x})$ (or $N(\mathbf{x})$) instead of iterating over all points. This neighborhood search step is helpful for finding a local minimum point. The proposed algorithm is summarized in Algorithm 1.

### 3.3 Convergence comparison: DPCD vs. SGM

One of the differences between the proposed DPCD and SGM [19] is the choice of $\mathbf{x}_i^k$ that are updated by Eq. (8) at the $k$-th iteration. SGM updates Eq. (8) for each $i$, while DPCD only changes the sign of $\mathbf{x}_i^k$ when the coordinate indexes are $L$-principal for some $L$. This difference is crucial to the convergences of the algorithms. More specifically, SGM can only guarantee convergence for the minimization of concave functions, while DPCD converges in finite steps for any functions with Lipschitz continuous gradients. We show the superiority of DPCD by the following example. The case $n = 2$ is presented in Fig. 2.

**Lemma 1.** *Consider the problem*

$$\min_{\mathbf{x} \in \{\pm 1\}^n} f(\mathbf{x}) := \min_{(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n) \in \{\pm 1\}^n} \frac{1}{2} \sum_{i=1}^n (\mathbf{x}_i + \beta_i)^2,$$

*where $0 < \beta_i < 1$. Let $\beta = \min_i \beta_i$. It holds that: (1) SGM generates a divergent sequence for any initial point; (2) With parameters $\alpha_1 = \alpha_2 = 1$ and $0 < \epsilon < \beta$, the proposed DPCD method always converges to the optimal solution.*

### 3.4 Theoretical convergence results

For simplicity, we ignore the neighborhood search part in the convergence analysis. Actually, the neighborhood search does not have any influence on the convergence since it always generates some binary vectors without increasing the value of the loss function. Thus, we can simply focus on the principal coordinate update part of the proposed DPCD method. We derive the following convergence results. When we say the algorithm converges in $T$ steps, we mean that the binary vector $\mathbf{x}^{T+1}$ obtained in the $(T+1)$-th iteration equals $\mathbf{x}^T$ in the $T$-th iteration (then the algorithm can stop here). Moreover, it is easy to see that our algorithm can converge to some local optimum with the help of the neighborhood search (without the neighborhood search, it only converges to some fixed binary vectors).

**Theorem 1.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a differentiable function such that $\nabla f$ is $L_0$-Lipschitz continuous on $[-1, 1]^n$. Setting the thresholds $L_1 = L_2 = L_0 + \epsilon$ where $\epsilon > 0$, and ignoring the neighborhood search. Then, the sequence $\mathbf{x}^k$ generated by Algorithm 1 always converges in $\frac{f_{max} - f_{min}}{2\epsilon}$ steps at most, where $f_{max} := \max_{\mathbf{x} \in \{\pm 1\}^n} f(\mathbf{x})$ and $f_{min} := \min_{\mathbf{x} \in \{\pm 1\}^n} f(\mathbf{x})$.*

The above theorem is very versatile since most loss functions in practice are $L_0$-Lipschitz continuous on $[-1, 1]^n$ for some $L_0 \in \mathbb{R}^+$. Furthermore, since $n$ is finite, $\{\pm 1\}^n$ is a finite set, thus the above $f_{max}$ and $f_{min}$ always exist and are finite. Furthermore, when $f(\mathbf{x})$ is quadratic, we have:

**Corollary 1.** *Let $f(\mathbf{x}) = \mathbf{x}^\top \mathbf{A} \mathbf{x} + \mathbf{c}^\top \mathbf{x} + d$ be some quadratic function where $\mathbf{A} \in \mathbb{R}^{n \times n}$, $\mathbf{x}, \mathbf{c} \in \mathbb{R}^n$, and $d \in \mathbb{R}$, in Theorem 1. Then, Algorithm 1 always converges in $\frac{\|\mathbf{A}\|_1 + \|\mathbf{c}\|_1}{\epsilon}$ steps at most.*

## 4 Experiments

In this section, we compare the DPCD algorithm with several state-of-the-art methods on two binary optimization tasks:

Table 1: Statistics for the graphs used in the dense subgraph discovery experiments.

| Graph | # Nodes | # Arcs | # Arcs/# Nodes |
|---|---|---|---|
| uk-2007-05 | 100000 | 3050615 | 30.506 |
| dblp-2010 | 326186 | 1615400 | 4.952 |
| eswiki-2013 | 972933 | 23041488 | 23.683 |
| hollywood-2009 | 1139905 | 113891327 | 99.913 |

Table 2: CPU time comparison (seconds) of dense subgraph discovery on the four graphs with $k = 1600$.

| Method | uk-2007-05 | dblp-2010 | eswiki-2013 | hollywood-2009 |
|---|---|---|---|---|
| DPCD | **1.77** | **6.19** | **14.17** | **21.80** |
| LP | 5.02 | 7.27 | 58.96 | 115.38 |
| RAVI | 2.28 | 6.40 | 104.94 | 77.95 |
| L2box-ADMM | 35.84 | 74.54 | 335.83 | 610.72 |
| MPEC-EPM | 45.21 | 158.11 | 1454.07 | 2114.20 |
| MPEC-ADM | 36.76 | 81.64 | 421.35 | 720.66 |

dense subgraph discovery and binary hashing. All codes are implemented in MATLAB using a workstation with an Intel 8-core 2.6GHz CPU and 32GB RAM.

## 4.1 Dense subgraph discovery

**Optimization problem.** Dense subgraph discovery [26, 27, 38, 41] has many applications in graph mining, such as real-time story identification [3], finding correlated genes [42] and graph visualization [1]. Let $G$ be a given undirected weighted graph with $n$ nodes, and $k$ be a given positive integer such that $1 \leq k \leq n$. The aim is to find the maximum density subgraph (the subgraph with the maximal sum of edge weights) with cardinality $k$. Let $\mathbf{W} \in \mathbb{R}^{n \times n}$ be the symmetric adjacency matrix of the graph $G$, where $\mathbf{W}_{ij}$ denotes the weight of the edge between vertices $i$ and $j$. Then the optimization problem can be formulated as:

$$\max_{\mathbf{x} \in \{0,1\}^n} \ \mathbf{x}^\top \mathbf{W} \mathbf{x}, \qquad \text{s.t. } \mathbf{x}^\top \mathbf{1} = k. \tag{9}$$

Note that, in this case, the variables are 0 or 1 instead of $-1$ or 1. In order to translate the problem to the form in Problem (2), the substitution $\mathbf{x} = \frac{1}{2}(\mathbf{y} + \mathbf{1})$ is adopted. Therefore, Problem (9) is equivalent to:

$$\min_{\mathbf{y} \in \{-1,1\}^n} \ -\mathbf{y}^\top \mathbf{W} \mathbf{y} - 2\mathbf{y}^\top \mathbf{W} \mathbf{1} - \mathbf{1}^\top \mathbf{W} \mathbf{1}, \qquad \text{s.t. } \mathbf{y}^\top \mathbf{1} = 2k - n. \tag{10}$$

In this way, the above problem can be approximately solved using our Algorithm 1.

**Graph datasets.** The experiments for dense subgraph discovery are conducted on four large-scale graphs *uk-2007-05, dblp-2010, eswiki-2013*, and *hollywood-2009* from The Laboratory for Web Algorithmics[1]. Table 1 gives a brief description of each graph. For example, hollywood-2009 contains roughly 1.13 million nodes and 113 million edges.

**DPCD vs. state-of-the-art methods.** The proposed DPCD method is compared with the methods LP [10], RAVI [26], L2box-ADMM [36], MPEC-EPM and MPEC-ADM [38]. The cardinality $k$ is chosen from the set $\{200, 400, 800, 1600, 3200, 6400\}$. For the DPCD method, we run one 5-neighborhood search ($m = 5$ in the neighborhood search setting) after ten principal coordinate updates, and set the maximum iteration number for the principal coordinate update part to 100. $L_1$ and $L_2$ are updated by Eq. (7). We tune the parameters $\alpha_1$ and $\alpha_2$ from $\{0.1, 0.2, 0.3, \ldots, 0.9, 1, 2, 3, \ldots, 10\}$ by cross-validation for each graph dataset. For other methods, we adopt the implementations and parameters suggested by the authors. The experimental results in Fig. 3 are reported in terms of $\mathbf{x}^\top \mathbf{W} \mathbf{x}/k$, which is the density of the subgraph with $k$ vertices. We can see that DPCD finds a denser subgraph than all compared methods in each case. Among the other methods, MPEC-EPM consistently outperforms LP in all the experiments. RAVI generally leads to solutions with low density. Furthermore, we provide CPU time comparisons for the six methods on the four graphs. From Table 2 we can see that the proposed DPCD achieves the fastest runtime on all graphs. This is due to the fast updates in Algorithm 1 (the main complexity comes from calculating the gradient of the loss function, which can be done quickly). Also, the efficiency of our method becomes more obvious as the graph size increases. LP and RAVI are faster than the other three methods, since MPEC-EPM needs to run the LP procedure multiple times, and L2-box ADMM and MPEC-ADM usually need more iterations to converge.

Table 3: Comparison between DPCD with and without the neighborhood search on values of the loss function and run time (in seconds). The loss function (9) is adopted. The cardinality of the dense subgraph is $k = 1600$.

| Method (Subgraph) | uk-2007-05 | dblp-2010 | eswiki-2013 | hollywood-2009 |
|---|---|---|---|---|
| DPCD (loss function) | **96.573** | **45.238** | **158.340** | **1944.921** |
| DPCD-0 (loss function) | 93.558 | 44.251 | 147.815 | 1866.240 |
| DPCD (run time) | 1.768 | 6.109 | 14.174 | 21.803 |
| DPCD-0 (run time) | **1.152** | **5.518** | **9.959** | **12.873** |

Table 4: Evaluation of DPCD and five general binary optimization methods with the same supervised loss function (12). The CIFAR-10 dataset is adopted. Results are reported in terms of MAP, Precision@500 and training time.

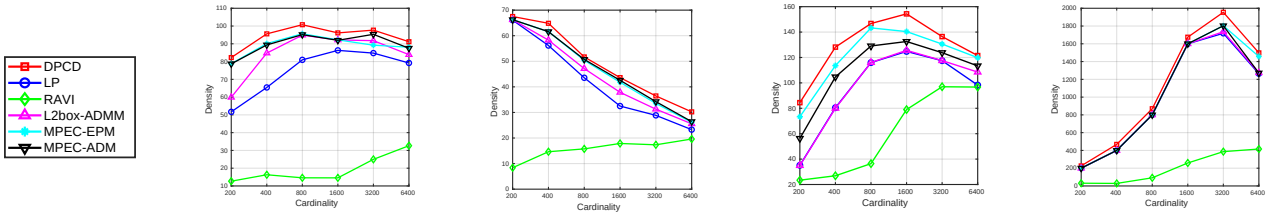
| Method | MAP | | | Precision@500 | | | Training time (seconds) | | |
|---|---|---|---|---|---|---|---|---|---|
| | 32 bits | 64 bits | 96 bits | 32 bits | 64 bits | 96 bits | 32 bits | 64 bits | 96 bits |
| DPCD | **0.7019** | **0.7088** | **0.7126** | **0.6337** | **0.6353** | **0.6370** | **3.76** | **6.01** | **9.39** |
| DCC | 0.5941 | 0.6193 | 0.6314 | 0.5486 | 0.5766 | 0.5894 | 11.18 | 36.03 | 158.87 |
| SGM | 0.6856 | 0.6986 | 0.7013 | 0.6177 | 0.6308 | 0.6360 | 7.87 | 10.83 | 16.30 |
| LP | 0.5237 | 0.5468 | 0.5459 | 0.4704 | 0.4972 | 0.4866 | 4.52 | 7.96 | 13.64 |
| L2box-ADMM | 0.6399 | 0.6724 | 0.6830 | 0.5929 | 0.6095 | 0.6162 | 43.07 | 90.10 | 200.94 |
| MPEC-EPM | 0.5823 | 0.6253 | 0.6276 | 0.5385 | 0.5738 | 0.5790 | 36.36 | 124.54 | 260.22 |

**With and without the neighborhood search.** We conduct a comparison of the proposed method with its variant without the neighborhood search technique, on the task of dense subgraph discovery with a cardinality of 1600 and in terms of loss functions and run time. DPCD-0 refers to our algorithm without neighborhood search. From Table 3, we can see that DPCD-0 is slightly faster, while DPCD usually achieves better loss functions.

## 4.2 Binary hashing

Binary hashing aims to encode high-dimensional data points, such as images and videos, into compact binary hash codes such that the similarities between the original data points and hash codes are preserved. This can be used to provide a constant or sub-linear search time and reduce the storage cost dramatically for such data points. The efficiency and effectiveness of binary hashing make it a popular tech-

[1] http://law.di.unimi.it/datasets.php




(a) uk-2007-05 (b) dblp-2010 (c) eswiki-2013 (d) hollywood-2009

Figure 3: Experimental results for dense subgraph discovery with $k \in \{200, 400, 800, 1600, 3200, 6400\}$. The metric is $\mathbf{x}^\top \mathbf{W}\mathbf{x}/k$, which can be seen as the density of the graph.

nique in machine learning, information retrieval and computer vision [8, 11, 19, 29, 31]. In a typical binary hashing task, $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)^\top \in \mathbb{R}^{n\times d}$ denotes a matrix of the original data points, where $\mathbf{x}_i \in \mathbb{R}^d$ is the $i$-th sample point, $n$ is the number of samples, and $d$ is the dimension of each sample. In the supervised setting, we let $\mathbf{Y} \in \mathbb{R}^{n\times c}$ be the label matrix, i.e., $\mathbf{Y}_{i,j} = 1$ if $\mathbf{x}_i$ belongs to the $j$-th class and 0 otherwise. The aim is to map $\mathbf{X}$ to some $\mathbf{B} = (\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n)^\top \in \{\pm 1\}^{n\times r}$, i.e., map each $x_i$ to a binary code $\mathbf{b}_i \in \{\pm 1\}^r$ for some small integer $r$ and preserve some similarities between the original data points in $\mathbf{X}$ and hash codes in $\mathbf{B}$. In the large-scale image retrieval tasks, we compute the Hamming distances between the hash codes of query images and database images, then use nearest neighbor search to find similar items for query images.

**Image datasets.** Three large-scale image datasets, CIFAR-10[2], ImageNet[3], and NUS-WIDE[4], are used in the binary hashing experiments. CIFAR-10 has 60k images, which are divided into 10 classes with 6k images each. We use a 384-dimensional GIST feature vector [24] to represent each image. 59k images are selected as the training set and the test set contains the remaining 1k images. A subset of ImageNet, ILSVRC 2012, contains 1.2 million images with 1k categories. As in [11, 28], we use a 4096-dimensional deep feature vector for each image, take 127K training images from the 100 largest classes, and 50K images from the validation set as the test set. NUS-WIDE contains 270K images with 81 labels. The images may have multiple labels. The 500-dimensional Bag-of-Words features are used here [7]. We adopt the 21 most frequent labels with the corresponding 193K images. For each label, 100 images are randomly selected as the test set and the remaining as the training set.

**DPCD vs. general binary optimization methods.** To illustrate the efficiency and effectiveness of our algorithm, we compare DPCD with several general state-of-the-art binary optimization methods DCC [28], SGM [19], LP [10], L2box-ADMM [36], and MPEC-EPM [38], on the image retrieval task. The dataset CIFAR-10 is adopted here. Various loss functions are designed for binary hashing (for examples, see Table 5). For fair comparison, we adopt the widely used supervised objective function [8, 28]

$$f(\mathbf{B}, \mathbf{W}) = \frac{1}{2}\|\mathbf{Y} - \mathbf{B}\mathbf{W}\|_2^2 + \frac{\delta}{2}\|\mathbf{W}\|_2^2 \qquad (11)$$

[2]http://www.cs.toronto.edu/kriz/cifar.html.

[3]http://www.image-net.org/.

[4]http://lms.comp.nus.edu.sg/research/NUS-WIDE.htm.


Figure 4: Top six retrieved neighbors of three query images, returned by DPCD, DCC, SGM, LP, L2box-ADMM and MPEC-EPM using 64 bits on CIFAR-10. The loss function (12) is adopted. The red background indicates false retrieved images.

for each method. Thus the optimization problem becomes:

$$\min_{\mathbf{B},\mathbf{W}} \frac{1}{2}\|\mathbf{Y} - \mathbf{B}\mathbf{W}\|_2^2 + \frac{\delta}{2}\|\mathbf{W}\|_2^2, \qquad (12)$$
$$\text{s.t.} \quad \mathbf{B} \in \{\pm 1\}^{n\times r}, \mathbf{W} \in \mathbb{R}^{r\times c},$$

where $\delta$ is a regularization parameter, and $\mathbf{W} \in \mathbb{R}^{r\times c}$ is the projection matrix (see [28]) which will be learned jointly with $\mathbf{B}$. The whole optimization runs iteratively over $\mathbf{B}$ and $\mathbf{W}$. When $\mathbf{W}$ is fixed, we apply the DPCD algorithm to $\mathbf{B}$. The key step is to calculate the gradient of $f(\mathbf{B}, \mathbf{W})$ as:

$$\nabla_{\mathbf{B}} f(\mathbf{B}, \mathbf{W}) = (\mathbf{B}\mathbf{W} - \mathbf{Y})\mathbf{W}^\top. \qquad (13)$$

Table 5: Comparison of three unsupervised methods (SADH-L, ARE and SGH) and three supervised methods (SDH, FSDH and FastHash) with/without using the proposed DPCD. ILSVRC 2012 and NUS-WIDE are adopted for unsupervised and supervised methods, respectively. Results are reported in terms of MAP, Precision@500 and training time. $\mathbf{S}$ = similarity matrix. $\mathbf{L}$ = Laplacian matrix of the similarity.

| Method | Loss Function | MAP | | | Precision@500 | | | Training time (s) | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 32 bits | 64 bits | 96 bits | 32 bits | 64 bits | 96 bits | 32 bits | 64 bits | 96 bits |
| SADH-L | $\min_{\mathbf{B}} \mathbf{Tr}(\mathbf{B}^\top \mathbf{L}\mathbf{B})$ | 0.2448 | **0.3104** | 0.3294 | 0.3692 | **0.4588** | 0.4835 | 121.90 | 384.01 | 738.33 |
| **DPCD** | | **0.2612** | 0.3085 | **0.3441** | **0.3945** | 0.4508 | **0.4992** | **19.03** | **33.50** | **48.29** |
| ARE | $\min_{\mathbf{B}} \|\mathbf{B}\mathbf{B}^\top - r\mathbf{X}\mathbf{X}^\top\|^2$ | 0.2509 | 0.2997 | 0.3276 | 0.3626 | 0.4478 | 0.4724 | 244.46 | 287.19 | 332.95 |
| **DPCD** | | **0.2808** | **0.3316** | **0.3607** | **0.3970** | **0.4856** | **0.5112** | **35.07** | **52.97** | **92.50** |
| SGH | $\min_{\mathbf{B}} \|\mathbf{B}\mathbf{B}^\top - r\mathbf{S}\|^2$ | **0.3225** | 0.4102 | 0.4367 | **0.4299** | 0.5220 | 0.5593 | 64.29 | 80.43 | 77.38 |
| **DPCD** | | 0.3148 | **0.4204** | **0.4523** | 0.4165 | **0.5252** | **0.5731** | **24.47** | **38.84** | **41.60** |
| SDH | $\min_{\mathbf{B},\mathbf{W}} \|\mathbf{Y} - \mathbf{B}\mathbf{W}\|^2 + \delta\|\mathbf{W}\|^2$ | 0.5716 | 0.5827 | 0.5920 | 0.6041 | 0.6056 | 0.6189 | 24.53 | 107.50 | 486.95 |
| **DPCD** | | **0.6124** | **0.6230** | **0.6392** | **0.6287** | **0.6357** | **0.6502** | **6.91** | **16.39** | **22.53** |
| FSDH | $\min_{\mathbf{B},\mathbf{W}} \|\mathbf{B} - \mathbf{Y}\mathbf{W}\|_2^2 + \delta\|\mathbf{W}\|_2^2$ | 0.5690 | 0.5639 | 0.5676 | 0.5918 | 0.5860 | 0.5988 | **2.76** | **3.07** | **5.95** |
| **DPCD** | | **0.6159** | **0.6170** | **0.6253** | **0.6260** | **0.6297** | **0.6335** | 5.42 | 8.96 | 10.99 |
| FastHash | $\min_{\mathbf{B}} \|\mathbf{B}\mathbf{B}^\top - r\mathbf{Y}\|^2$ | 0.5174 | 0.5398 | 0.5403 | 0.5867 | 0.6014 | 0.6180 | 1381.75 | 4668.13 | 10605.82 |
| **DPCD** | | **0.5621** | **0.5579** | **0.5767** | **0.5991** | **0.6145** | **0.6226** | **6.42** | **7.88** | **11.79** |

Then, $L_1, L_2$ can be obtained by Eq. (7). After deriving $S^{k+}$ and $S^{k-}$, we update $\mathbf{B}$ by Eq. (8). When $\mathbf{B}$ is fixed, $\mathbf{W}$ can be updated by

$$\mathbf{W} = \arg\min_{\mathbf{W}^*} f(\mathbf{B}, \mathbf{W}^*) = (\mathbf{B}^\top \mathbf{B} + \delta \mathbf{I}_r)^{-1} \mathbf{B}^\top \mathbf{Y}. \quad (14)$$

Finally, we adopt the linear hash function $h(\mathbf{X}) = \mathrm{sgn}(\mathbf{X}\mathbf{P})$ to encode $\mathbf{X}$ onto binary codes, where $\mathbf{P} \in \mathbb{R}^{d \times r}$ can be derived by:

$$\mathbf{P} = \arg\min_{\mathbf{P}^*} \|\mathbf{X}\mathbf{P}^* - \mathbf{B}\|^2 = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{B}. \quad (15)$$

During the test phase, for a query item, first we use the above linear hash function to derive the hash code, then adopt nearest neighborhood search under Hamming distance to find its similar items. The regularization parameter $\delta$ in Eq. (12) is set to 1. For the DPCD method, we run one 5-neighborhood search after 10 principal coordinate updates, tune the parameters $\alpha_1$ and $\alpha_2$ from $\{0.1, 0.2, 0.3, \ldots, 0.9, 1, 2, 3, \ldots, 10\}$ by cross-validation for each dataset and each binary code length, and set the maximum iteration number for $\mathbf{B}$ to 20 each time when $\mathbf{W}$ is fixed. We run at most five iterations for updating $\mathbf{B}$ and $\mathbf{W}$ iteratively. For other methods, we only use their algorithms to update $\mathbf{B}$. The updates of $\mathbf{W}$ are the same as DPCD. We adopt the implementations and parameters suggested by the authors for these methods. Ground truths are defined by the label information from the datasets. The experimental results are reported in terms of mean average precision (MAP), Precision@500 and training time efficiency (we ignore the comparison of test time here since the test parts are similar for each method). The experiments are conducted on the CIFAR-10 dataset. From Table 4 we can see that DPCD outperforms all other methods in MAP and Precision@500. For instance, on CIFAR-10 with 96 bits, DPCD outperforms DCC by 8.1%, and MPEC-EPM by 8.5% in terms of MAP. It is clear that increasing the number of bits yields better performance for all methods. Also, the training time for the proposed DPCD method is always faster than other compared methods. For example, DPCD runs 20 times faster than MPEC-EPM on 64 bits, which verifies that one major advantage of our method is the fast optimization process. Furthermore, for each of the above six methods, we give in Fig. 4 the top six retrieved neighbors of three query images, on the CIFAR-10 dataset with 64 binary bits, where the distance between two images is defined by the Hamming distance between their corresponding hashing binary bits. From Fig. 4 we can see that DPCD always returns images with same labels as the query images. LP returns several false images due to the relaxation. Other methods also retrieve several database images with different labels from the query images. Overall, we conclude that DPCD performs best among the compared methods in the image retrieval task.

**Algorithm complexity analysis.** Now we discuss the complexity of the above supervised DPCD algorithm. The gradient $\nabla_{\mathbf{B}} f(\mathbf{B}, \mathbf{W})$ is calculated in Eq. (13); thus the complexity is $O(nrc)$. Then $L_1$ and $L_2$ are derived by Eq. (7), which has a complexity of $O(nr)$, since there are $nr$ additions in Eq. (7). Furthermore, $S^{k+}$ and $S^{k-}$ can be calculated by the values of all $\nabla_{ij} f(\mathbf{B}^k)$ where $1 \le i \le n$, $1 \le j \le r$, which also has a complexity of $O(nr)$. Similarly, the update for $\mathbf{B}$ has a complexity of $O(nr)$. A neighborhood search has complexity $O(nrc)$ due to the calculation of $f(\mathbf{B}, \mathbf{W})$. Let $T$ be the maximum iteration number of principal coordinate update parts for updating $\mathbf{B}$. Then the total complexity of updating $\mathbf{B}$ is $O(Tnrc)$. When $\mathbf{B}$ is given, the complexity for updating $\mathbf{W}$ in Eq. (14) is $O(ncr + r^3 + cr^2)$. The complexity for calculating $\mathbf{P}$ in Eq. (15) is $O(d^3 + nd^2 + ndr)$. Suppose that there are at most $t$ iterations for updating $\mathbf{B}$ and $\mathbf{W}$ iteratively. Since $r, c \ll n$, the total complexity for supervised DPCD is $O(tTncr + d^3 + nd^2 + ndr)$, which is a linear function of $n$.

**DPCD vs. specific binary hashing methods.** DPCD is not only fast, but also very versatile, and can thus be used to handle many different loss functions. To demonstrate this, we apply DPCD to several widely used loss functions, and compare the results with the corresponding hashing methods that were designed for each specific loss function. The optimization process is similar to the supervised case. Table 5 illustrates a comparison of DPCD with three unsupervised methods, SADH-L [29], ARE [11], and SGH [12], on the ILSVRC 2012 dataset, and three supervised hashing methods, SDH [28], FSDH [8], and FastHash [16], on the NUS-WIDE dataset, using their specific loss functions. The proposed DPCD shows increased performance over the original methods and achieves higher MAP and Precision@500 in most cases, especially for the supervised loss functions. In terms of the training time, DPCD outperforms all methods except FSDH (FSDH also runs very fast due to the closed form updating at each iteration). The proposed DPCD can significantly decrease the training time for unsupervised loss functions due to the fast updating of the binary codes $\mathbf{B}$. Finally, we conclude that DPCD is a fast and effective optimization method for large-scale image retrieval tasks.

## 5 Conclusion

This paper presents a novel fast optimization method, called Discrete Principal Coordinate Descent (DPCD), to approximately solve binary optimization problems with/without restrictions on the numbers of 1s and $-1$s in the variables. We derive several theoretical results on the convergence of the proposed algorithm. Experiments on dense subgraph discovery and binary hashing demonstrate that our method generally outperforms state-of-the-art methods in terms of both solution quality and optimization efficiency.

## 6 Acknowledgements

We really appreciate the valuable suggestions given by reviewers for improving the overall quality of this paper. We also would like to thank Dr. Mengyang Yu for helpful discussions.

## References

[1] Alvarez-Hamelin, J. I.; Dall'Asta, L.; Barrat, A.; and Vespignani, A. 2006. Large scale networks fingerprinting and visualization using the k-core decomposition. In *Advances in Neural Information Processing Systems (NIPS)*, 41–50.

[2] Ames, B. P. 2015. Guaranteed recovery of planted cliques and dense subgraphs by convex relaxation. *Journal of Optimization Theory and Applications*, 167(2): 653–675.

[3] Angel, A.; Sarkas, N.; Koudas, N.; and Srivastava, D. 2012. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *Proceedings of the VLDB Endowment*, 5(6): 574–585.

[4] Balalau, O. D.; Bonchi, F.; Chan, T.; Gullo, F.; and Sozio, M. 2015. Finding subgraphs with maximum total density and limited overlap. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, 379–388. ACM.

[5] Bi, S.; Liu, X.; and Pan, S. 2014. Exact penalty decomposition method for zero-norm minimization based on MPEC formulation. *SIAM Journal on Scientific Computing*, 36(4): A1451–A1477.

[6] Boyd, S.; Parikh, N.; Chu, E.; Peleato, B.; Eckstein, J.; et al. 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1): 1–122.

[7] Chua, T.-S.; Tang, J.; Hong, R.; Li, H.; Luo, Z.; and Zheng, Y.-T. July 8-10, 2009. NUS-WIDE: A Real-World Web Image Database from National University of Singapore. In *Proc. of ACM Conf. on Image and Video Retrieval (CIVR'09)*. Santorini, Greece.

[8] Gui, J.; Liu, T.; Sun, Z.; Tao, D.; and Tan, T. 2018. Fast supervised discrete hashing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(2): 490–496.

[9] He, L.; Lu, C.-T.; Ma, J.; Cao, J.; Shen, L.; and Yu, P. S. 2016. Joint community and structural hole spanner detection via harmonic modularity. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 875–884. ACM.

[10] Hsieh, C.-J.; Natarajan, N.; and Dhillon, I. S. 2015. PU Learning for Matrix Completion. In *ICML*, 2445–2453.

[11] Hu, M.; Yang, Y.; Shen, F.; Xie, N.; and Shen, H. T. 2018. Hashing with angular reconstructive embeddings. *IEEE Transactions on Image Processing*, 27(2): 545–555.

[12] Jiang, Q.-Y.; and Li, W.-J. 2015. Scalable graph hashing with feature transformation. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

[13] Johnson, D. S.; and Garey, M. R. 1979. *Computers and intractability: A guide to the theory of NP-completeness*, volume 1. WH Freeman San Francisco.

[14] Komodakis, N.; and Tziritas, G. 2007. Approximate labeling via graph cuts based on linear programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(8): 1436–1453.

[15] Li, G.; and Pong, T. K. 2015. Global convergence of splitting methods for nonconvex composite optimization. *SIAM Journal on Optimization*, 25(4): 2434–2460.

[16] Lin, G.; Shen, C.; Shi, Q.; Van den Hengel, A.; and Suter, D. 2014. Fast supervised hashing with decision trees for high-dimensional data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1963–1970.

[17] Lin, G.; Shen, C.; Suter, D.; and Van Den Hengel, A. 2013. A general two-step approach to learning-based hashing. In *Proceedings of the IEEE international conference on computer vision*, 2552–2559.

[18] Liu, L.; Shao, L.; Shen, F.; and Yu, M. 2017. Discretely coding semantic rank orders for supervised image hashing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1425–1434.

[19] Liu, W.; Mu, C.; Kumar, S.; and Chang, S.-F. 2014. Discrete graph hashing. In *Advances in Neural Information Processing Systems (NIPS)*, 3419–3427.

[20] Lu, Z.; and Zhang, Y. 2013. Sparse approximation via penalty decomposition methods. *SIAM Journal on Optimization*, 23(4): 2448–2478.

[21] Luo, Y.; Yang, Y.; Shen, F.; Huang, Z.; Zhou, P.; and Shen, H. T. 2018. Robust discrete code modeling for supervised hashing. *Pattern Recognition*, 75: 128–135.

[22] Mehrotra, S. 1992. On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization*, 2(4): 575–601.

[23] Murray, W.; and Ng, K.-M. 2010. An algorithm for nonlinear optimization problems with binary variables. *Computational Optimization and Applications*, 47(2): 257–288.

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

[25] Olsson, C.; Eriksson, A. P.; and Kahl, F. 2007. Solving large scale binary quadratic problems: Spectral methods vs. semidefinite programming. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, 1–8. IEEE.

[26] Ravi, S. S.; Rosenkrantz, D. J.; and Tayi, G. K. 1994. Heuristic and special case algorithms for dispersion problems. *Operations Research*, 42(2): 299–310.

[27] Rozenshtein, P.; Bonchi, F.; Gionis, A.; Sozio, M.; and Tatti, N. 2018. Finding events in temporal networks:

Segmentation meets densest-subgraph discovery. In *2018 IEEE International Conference on Data Mining (ICDM)*, 397–406. IEEE.

[28] Shen, F.; Shen, C.; Liu, W.; and Tao Shen, H. 2015. Supervised discrete hashing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 37–45.

[29] Shen, F.; Xu, Y.; Liu, L.; Yang, Y.; Huang, Z.; and Shen, H. T. 2018. Unsupervised deep hashing with similarity-adaptive and discrete optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[30] Shi, X.; Ling, H.; Xing, J.; and Hu, W. 2013. Multi-target tracking by rank-1 tensor approximation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2387–2394.

[31] Wang, J.; Zhang, T.; Sebe, N.; Shen, H. T.; et al. 2018. A survey on learning to hash. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4): 769–790.

[32] Wang, M.; and Sha, F. 2011. Information theoretical clustering via semidefinite programming. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 761–769.

[33] Wang, P.; Shen, C.; van den Hengel, A.; and Torr, P. H. 2017. Large-scale binary quadratic optimization using semidefinite relaxation and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(3): 470–485.

[34] Wang, Y.; Yin, W.; and Zeng, J. 2015. Global convergence of ADMM in nonconvex nonsmooth optimization. *Journal of Scientific Computing*, 1–35.

[35] Wright, S. J. 2015. Coordinate descent algorithms. *Mathematical Programming*, 151(1): 3–34.

[36] Wu, B.; and Ghanem, B. 2018. lp-box ADMM: A versatile framework for integer programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[37] Xiong, H.; Yu, M.; Liu, L.; Zhu, F.; Qin, J.; Shen, F.; and Shao, L. 2021. A Generalized Method for Binary Optimization: Convergence Analysis and Applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[38] Yuan, G.; and Ghanem, B. 2016. Binary Optimization Via Mathematical Programming With Equilibrium Constraints. *arXiv preprint arXiv:1608.04425*.

[39] Yuan, G.; and Ghanem, B. 2016. Sparsity constrained minimization via mathematical programming with equilibrium constraints. *arXiv preprint arXiv:1608.04430*.

[40] Yuan, G.; and Ghanem, B. 2017. An Exact Penalty Method for Binary Optimization Based on MPEC Formulation. In *AAAI*, 2867–2875.

[41] Yuan, X.-T.; and Zhang, T. 2013. Truncated power method for sparse eigenvalue problems. *Journal of Machine Learning Research*, 14(Apr): 899–925.

[42] Zhang, B.; and Horvath, S. 2005. A general framework for weighted gene co-expression network analysis. *Statistical applications in genetics and molecular biology*, 4(1).

[43] Zhang, Z.; Li, T.; Ding, C.; and Zhang, X. 2007. Binary matrix factorization with applications. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, 391–400. IEEE.