# Optimized Distances for Binary Code Ranking

Jianfeng Wang[1] Heng Tao Shen[2] Shuicheng Yan[3]
Nenghai Yu[1] Shipeng Li[4] Jingdong Wang[4]

[1] University of Science and Technology of China, Hefei, P. R. China
[2] The University of Queensland, Australia
[3] National University of Singapore, Singapore
[4] Microsoft Research, Beijing, P. R. China

wjf2006@mail.ustc.edu.cn; shenht@itee.uq.edu.au;eleyans@nus.edu.sg;
ynh@ustc.edu.cn; {spli;jingdw}@microsoft.com

## ABSTRACT

Binary encoding on high-dimensional data points has attracted much attention due to its computational and storage efficiency. While numerous efforts have been made to encode data points into binary codes, how to calculate the effective distance on binary codes to approximate the original distance is rarely addressed. In this paper, we propose an effective distance measurement for binary code ranking. In our approach, the binary code is firstly decomposed into multiple sub codes, each of which generates a query-dependent distance lookup table. Then the distance between the query and the binary code is constructed as the aggregation of the distances from all sub codes by looking up their respective tables. The entries of the lookup tables are optimized by minimizing the misalignment between the approximate distance and the original distance. Such a scheme is applied to both the symmetric distance and the asymmetric distance. Extensive experimental results show superior performance of the proposed approach over state-of-the-art methods on three real-world high-dimensional datasets for binary code ranking.

## Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Search Process

## General Terms

Algorithms, Measurement, Experimentation

## Keywords

Binary Code Ranking; Lookup Table; Approximate Nearest Neighbor Search

## 1. INTRODUCTION

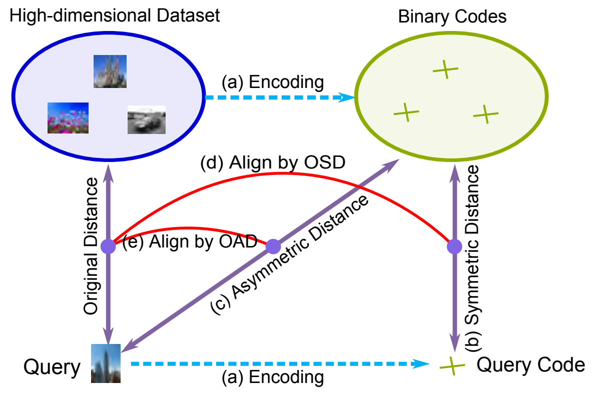


**Figure 1: Hashing-based ANN search. (a) encoding of the high-dimensional data to binary codes; (b) distance between binary codes; (c) distance between unbinarized query and the binary code; (d) alignment of the symmetric distance towards the original distance by our OSD; (e) alignment of the asymmetric distance towards the original distance by our OSD.**

Nearest neighbor (NN) search is a fundamental problem in many multimedia and computer vision applications. Due to the scalability issue and "curse of dimensionality", exact NN search is often intractable for large-scale high-dimensional datasets in practice. Instead, approximate nearest neighbor (ANN) search can provide orders of magnitude faster speed with near-optimal accuracy and is thus more widely applied in real applications than exact NN search [21].

Recently, hashing for ANN search has attracted lots of attention because of its storage and computational efficiency. High-dimensional data points are represented as compact binary codes, and the distance on binary codes is adopted as the approximate measure towards the original distance[1]. Numerous efforts have been devoted to encode the points into binary codes. One representative approach is locality-sensitive hashing (LSH) [6] which employs random projections to compute the binary codes. It is guaranteed that similar data points are mapped into the same binary codes with

[1]Here we assume that the original distance is computed by the Euclidean distance in the high-dimensional data space.

a high probability. Instead of random projection-based approaches, one can learn the hash functions from the databases by preserving the similarities evaluated from the original space in the binary code space [8, 15, 16, 18, 22, 27, 28, 29, 30, 34, 35]. The semantic information can be incorporated to construct the hash functions, which is not our focus. In this paper, we mainly target on the search based on the Euclidean distance. Fig. 1 (a) shows the role of hashing functions in the search pipeline.

After obtaining binary codes, the distance between the query and the code is then used to rank all the database points. The Hamming distance, which counts the number of bits that differ between two binary codes, is mostly employed in existing works [3, 16, 27, 29]. It can be quickly calculated with several hardware instructions. However, its effectiveness is limited, and only $Q + 1$ different values are provided, where $Q$ is the number of bits (i.e., code length). As a consequence, for a query point, many database points may share the same Hamming distance value.

Besides encoding the query into the binary code, another way is to directly define some appropriate distance between the unencoded query and the database binary code, which is usually referred to as *asymmetric distance* (AD) [4]. Comparatively, the distance using the encoded query is referred to as *symmetric distance* (SD). Fig. 1 (b) and (c) illustrate the relations of SD and AD, respectively. Since the unencoded query carries more information than the encoded query, AD generally achieves better accuracy than SD. In [4, 31, 33], query-dependent weights are computed for each bit under some assumptions, such as the uniform distribution assumption [33], and the distance is defined as the summation of a portion of the weights according to the binary codes. For product quantization [8], and composite quantization [32] the space is grouped into multiple clusters. Each database point is encoded by the index of the nearest cluster center and AD is defined as the distance between the query and the cluster center.

Given a query point and binary codes, in this paper we study the problem of *how to effectively measure the distance between the query and the database binary codes for the ANN search*. Here the effectiveness refers to the alignment between the search results based on the approximate distance and those based on the original distance. Suppose the code length is $Q$, there are in total of $2^Q$ different possible codes. A simple but effective way is to build up a lookup table with $2^Q$ entries, each of which represents the distance between the query and the corresponding binary code. Obviously, such a table can contain complete distance information. By carefully setting the values in the lookup table, we expect a much better approximate distance as well as ranking result.

However, when the code length $Q$ increases, storing $2^Q$ values in the table becomes infeasible. To address this issue, we propose to partition the $Q$ bits into multiple sub codes, each of which generates a much smaller lookup table. Then, the overall approximate distance is defined as the aggregation from these lookup tables of all sub codes. We optimize the entry values in the lookup tables by minimizing the misalignment between the approximate distance and the original distance. We term the proposed idea applied to SD as *Optimized Symmetric Distance* (OSD) while to AD as *Optimized Asymmetric Distance* (OAD). Fig. 1 (d) and (e) depict the alignment of our OSD and OAD, respectively. Extensive experimental results demonstrate the superior effectiveness of our approach over state-of-the-art methods for binary code ranking.

Our approach has the following advantages. 1) It generalizes the existing approaches and achieves better ranking results. 2) It can be applied to a wide range of binary codes, including the codes generated by product quantization [8]. 3) The algorithm makes no specific assumption on the data distribution (e.g. uniform distribution). 4) It is efficient and consumes linear training time to the data size. The extra testing time is minor and is consistent with the data size.

## 2. RELATED WORK

In this section, we review hashing-based ANN search algorithms that are closely related to our approach. The hashing-based search algorithms consist of two main phases. The first phase is binary code encoding, in which each data point is encoded as a binary code. The second phase is binary code ranking, in which all data points are ranked based on their distances computed on binary codes to the query.

### 2.1 Binary code encoding

There are generally two types of approaches towards encoding high-dimensional data points into binary codes. One is based on random projections and the other is to learn from the data.

Locality sensitive hashing (LSH) [6] is one of the typical random-based approaches. The key idea is to hash high-dimensional data points using several random hash functions to ensure that the probability of collision is high for close points and vice versa. There are various algorithms extending LSH, e.g. Kernelized LSH [13], Shift-invariant kernel hashing [20], Super-bit LSH [9]. These approaches are blind with the data distribution and usually need longer binary codes or more hash tables to achieve a satisfactory performance.

In contrast, learning-based algorithms compute the hash function explicitly from the training data. Binary reconstructive embedding [12] aims to align the Hamming distance to the original Euclidean distance. Iterative quantization hashing [3] and isotropic hashing [10] are based on PCA and rotate the axis by minimization of the distortion errors and by the equality of variances, respectively. Spectral hashing (SH) [29] learns the hash codes by minimizing the summation of the Hamming distances weighted by the similarities computed in the input space, which is extended from different perspectives in [17, 28]. The sequential projection learning scheme [25] and LDAHash [23] learn hash functions to minimize (maximize) the Hamming distance of similar (dissimilar) pairs. Ranking-based hashing approaches are developed in [15, 27] to expect the ranking order is well aligned from the binary space to the original space.

Product quantization (PQ) [8] encodes the points by the index of the nearest cluster center with the quantization errors minimized. The training time cost for a large number of clusters is reduced through partitioning the original space into multiple disjoint subspaces and clustering each subspace. The quantization errors are further reduced in [2, 18, 26, 32].

### 2.2 Binary code ranking

Given a query point, its distance to the binary code of each database point is typically used as the approximation of the true distance in the original high-dimensional space. Due to the fast speed of the approximate distance computation, even the exhaustive search over the whole dataset is fast enough in practice. Sophisticated techniques based on the inverted index can further improve the speed, e.g., [1, 8, 19]. In this phase, the approximate distance plays a crucial role on the overall ranking performance.

The distance approximation scheme consists of two groups: symmetric distance (SD) and asymmetric distance (AD). SD is referred to as the distance between the data from the same space while AD is referred to as the distance between the data from different spaces.

**Symmetric Distance:** The Hamming distance is most widely used and can be computed efficiently by bit operations. However, the Hamming distance is often not sufficiently effective and many retrieved candidates may share the same distance value. The Spherical Hamming distance [5] is based on the hypersphere hashing scheme and is defined as the Hamming distance divided by the number of bits where the points falls into the same hypersphere. Manhattan hashing [11] uses the Manhattan distance with each projected dimension encoded by natural binary codes. The weighted Hamming distance, where each bit is assigned to a real-valued weight, is used in [15, 28] to rank the database points. For product quantization [8] and cartesian $K$-means [18], the space is split into multiple clusters, and the distance between the cluster centers is used to approximate the original distance.

**Asymmetric Distance:** AD does not encode the query point as the binary code and thus is able to exploit more accurate query representation. The distance in [31] assigns a bit-level weight, which depends on the *unbinarized hash values* [31] of the query and the database points. Two kinds of ADs are proposed in [4], based on the assumption that the Euclidean distance between the unbinarized hash values is closely related with the original distance. QsRank [33] defines the query-sensitive distance for the PCA-based binary codes. The asymmetric Hamming distance [7] is incorporated within the inverted file structure. In [8, 18, 32], the distance between the original query point and the cluster centers is used to approximate the original distance with each database point encoded as the index of the nearest cluster.

In this paper, we focus on the second phase, and aim to define the optimized distances on binary codes to achieve effective binary code ranking. The first phase to encode the high-dimensional data is not our contribution.

## 3. OVERVIEW

Given a dataset $\mathcal{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ with $\mathbf{x}_i \in \mathbb{R}^P$ where $N$ is the dataset size and $P$ is the dimensionality of data space, the encoding scheme encodes each data point into a binary code, and can be written as $\mathbf{h} : \mathbb{R}^P \mapsto \{0,1\}^Q$, where $Q$ is the code length. Let $\mathbf{b}_i \in \{0,1\}^Q$ be the binary code of $\mathbf{x}_i$. Provided the high-dimensional dataset and the derived binary codes, we aim to optimize the distance on binary codes to improve the search accuracy.

Given a query point $\mathbf{q} \in \mathbb{R}^P$, the distance comparison is mainly determined by the binary codes of the database points. Since $Q$ bits lead to $2^Q$ different codes, we can build up a lookup table with $2^Q$ entries, each of which represents the distance between the query and the corresponding binary code. Fig. 2 (a) illustrates the lookup table with $2^{16}$ entries when $Q = 16$.

**Table 1: Sizes of lookup tables with different numbers of partitions when $Q = 32$.**

| $T$ | 1 | 2 | 4 | 8 | 16 |
|---|---|---|---|---|---|
| $T \times 2^{Q/T}$ | $2^{32}$ | $2^{17}$ | $2^{10}$ | $2^7$ | $2^6$ |

However, with the code length $Q$ increased, storing $2^Q$ entries becomes infeasible in practice. To address the scalability problem, we propose to partition the $Q$ bits into multiple sub codes, and each sub code generates a much smaller lookup table. Let $T$ be the number of partitions. For the sake of simplicity, we assume $Q$ is divisible by $T$ and the length of each sub code is equal to $S \triangleq Q/T$, where the symbol '$\triangleq$' means 'defined as'. Our approach can be easily applied to partitions with an unequal length. Since the length of the sub code is $S$, there are $M \triangleq 2^S$ possible binary codes or *buckets* on each partition. Denote the smaller lookup table as $\mathbf{d}^t(\mathbf{q}) \in \mathbb{R}^M, t \in \{1, \cdots, T\}$ for the query $\mathbf{q}$. In certain cases, we eliminate $\mathbf{q}$ and use $\mathbf{d}^t$ to denote the lookup table without confusion. Fig. 2 (b) shows 2 smaller lookup tables when $T = 2$ and $Q = 16$.

Before describing how to aggregate the distance from the lookup tables $\{\mathbf{d}^t\}$, we introduce some notations. Let $\mathbf{b}_i^t \in \{0,1\}^S$ be the $t$-th sub code of the binary code $\mathbf{b}_i$. Given $\mathbf{b}_i^t$, we further introduce an indicator matrix $\mathbf{R}^t \in \{0,1\}^{N \times M}$, in which the entry $r_{j,m}^t$ on the $j$-th row and $m$-th column is defined as

$$r_{j,m}^t \triangleq \begin{cases} 1, & m = \mathcal{P}(\mathbf{b}_j^t) \\ 0, & \text{otherwise,} \end{cases} \qquad (1)$$

where $\mathcal{P}(\mathbf{b}_j^t)$ maps the binary representation $\mathbf{b}_j^t$ into the natural decimal representation. For example, $\mathcal{P}(0100) = 4$ and $\mathcal{P}(1001) = 9$. In other words, the condition $m = \mathcal{P}(\mathbf{b}_j^t)$ means that on the $t$-th partition, the $j$-th point falls into the $m$-th bucket. The two representations, $\mathbf{b}_j^t$ and $r_{j,m}^t$ are equivalent, and we can easily derive one from the other.

Then, we arrive at the proposed distance between the query $\mathbf{q}$ and the binary code $\mathbf{b}_j$ with $T$ partitions,

$$\mathcal{D}(\mathbf{q}, \mathbf{b}_j) \triangleq \sum_{t,n} r_{j,n}^t d_n^t(\mathbf{q}), \qquad (2)$$

where $d_n^t(\mathbf{q})$ is the $n$-th entry of $\mathbf{d}^t(\mathbf{q})$. The intuition of the distance definition is to select one entry from each $\mathbf{d}^t$ and aggregate these selected entries. With fixed $t$, the summation through all $n$ equivalently extracts the entry $d_{n_t}^t$, where on the $t$-th partition the $j$-th point falls into the $n_t$-th bucket. Then over all the possible $t$ values, we sum up $d_{n_t}^t$ to obtain the overall distance.

If no partitioning is performed, i.e. $T = 1$, the lookup table $\{d_n^1\}$ contains $2^Q$ entries and each entry denotes the distance to the corresponding binary code. When $T > 1$, the distance is constructed as the summation of $T$ entries. By partitioning, the size of lookup tables reduces to $T \times 2^{Q/T}$ from $2^Q$. It can be easily proved that with $T \leq Q/2$, the size is strictly decreased with a larger $T$. Table 1 illustrates the sizes with different numbers of partitions when $Q = 32$.

Then, all the database points can be ranked by the distance defined in Eqn. 2. In Sec. 4 and Sec. 5, we will present the proposed approaches to learn $\{\mathbf{d}^t(\mathbf{q})\}$ for SD and AD.

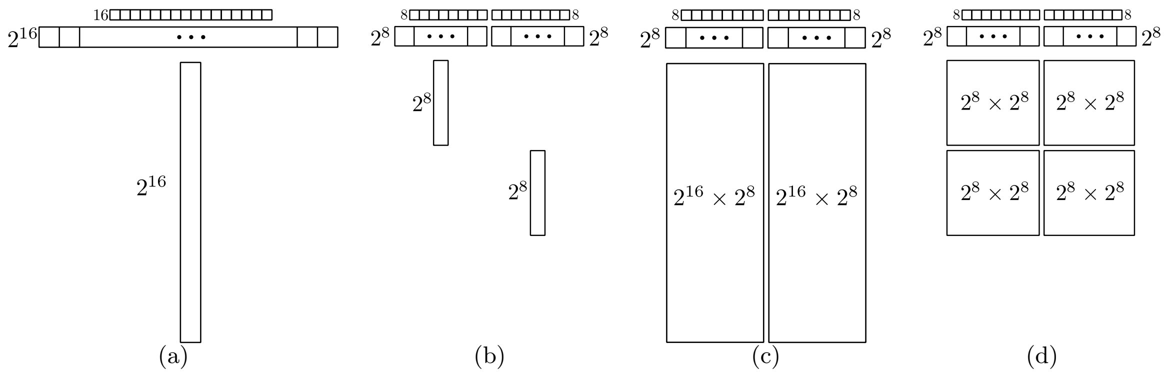


**Figure 2: Proposed lookup tables for binary code ranking with 16 bits. (a) a query-dependent lookup table to indicate the distance to any binary code of the database. (b) 2 smaller lookup tables by 2 partitions on the database code. (c) 2 lookup matrices to store all the distance information between the query code and the database code. (d) 4 smaller lookup matrices by further splitting the query code into 2 groups.**

## 4. OPTIMIZED SYMMETRIC DISTANCE

For symmetric distance, the query point is encoded as a binary code, denoted by $\mathbf{b}_q$. Accordingly, the lookup table $\{\mathbf{d}^t(\mathbf{q})\}$ is written as $\{\mathbf{d}^t(\mathbf{b}_q)\}$. Since there are $2^Q$ different possible query codes and $2^S$ different codes for each of the $T$ partitions of the database codes, we can imagine to prebuild $T$ lookup matrices with the size $2^Q \times 2^S$. Fig. 2 (c) illustrates the lookup matrix with the size $2^{16} \times 2^8$ when $T = 2$ and $Q = 16$. During the search stage, we can extract the corresponding row as the lookup table $\mathbf{d}^t(\mathbf{b}_q)$.

However, storing the matrix with the size $2^Q \times 2^S$ is infeasible. To address the problem, we propose to partition the query code in the same way as done for the database code $\mathbf{b}_j$. Let $\mathbf{D}^{s,t} \in \mathbb{R}^{M\times M}$ be the lookup matrix between the $s$-th partition of the query code and the $t$-th partition of the database code, and $d^{s,t}_{m,n}$ be the entry on the $m$-th row and the $n$-th column of $\mathbf{D}^{s,t}$. Fig. 2 (d) shows the example when $T = 2$ and $Q = 16$. Then, the element of the lookup tables is defined as

$$d^t_n(\mathbf{q}) \triangleq \sum_{s,m} r^s_{q,m} d^{s,t}_{m,n}, \tag{3}$$

where $r^s_{q,m}$ is the indicator notation of the query code, similar as in Eqn. 1 for the database code. If $r^s_{q,m_s}$ is non-zero, Eqn. 3 is equivalent to $d^t_n = \sum_s r^s_{q,m_s} d^{s,t}_{m_s,n}$. That is, every partition of the query code contributes one entry to construct $d^t_n$.

The lookup matrix $\{\mathbf{D}^{s,t}\}$ contains all the distances between any query code to any database codes. We offline learn $\{\mathbf{D}^{s,t}\}$ from the training data, and compute the lookup table $\{\mathbf{d}^t(\mathbf{b}_q)\}$ for the query code $\mathbf{b}_q$ by Eqn. 3.

### 4.1 Formulation

The original distance between the query point $\mathbf{q}$ and the database point $\mathbf{x}_j$ is

$$\|\mathbf{q} - \mathbf{x}_j\|_2, \tag{4}$$

while the approximate symmetric distance between their binary codes is constructed by Eqn. 2 and Eqn. 3. Obviously, there may exist a distance approximation error, and it is beneficial to minimize it for accurate ANN search. Substituting Eqn. 3 into Eqn. 2, we propose to learn the values $\{d^{s,t}_{m,n}\}$ by minimizing

$$\mathrm{E}_{\mathrm{q}}\left(\sum_j (\|\mathbf{q} - \mathbf{x}_j\|_2^2 - \sum_{t,n} r^t_{j,n} \sum_{s,m} r^s_{q,m} d^{s,t}_{m,n})^2\right), \tag{5}$$

where $\mathrm{E}_{\mathrm{q}}(\cdot)$ represents the expectation over the distribution of the query $\mathbf{q}$.

Assuming that the query and the database points follow the same data distribution, the problem of minimizing Eqn. 5 is transformed to

$$\min_{\{d^{s,t}_{m,n}\}} \mathcal{F}_{\mathrm{S}} \triangleq \frac{1}{2}\sum_{i,j} (\|\mathbf{x}_i - \mathbf{x}_j\|_2^2 - \sum_{s,t,m,n} r^s_{i,m} r^t_{j,n} d^{s,t}_{m,n})^2. \tag{6}$$

In this formulation, $d^{s,t}_{m,n}$ is not required to be non-negative. The only requirement is the overall approximate distance $\sum_{s,t,m,n} r^s_{i,m} r^t_{j,n} d^{s,t}_{m,n}$ should align as well as possible to the squared original distance $\|\mathbf{x}_i - \mathbf{x}_j\|_2^2$. Here we use the squared Euclidean distance rather than the Euclidean distance because of the fact that the squared distance can be exactly written as the summation of multiple values.

### 4.2 Optimization

The matrix form of Eqn. 6 is

$$\mathcal{F}_{\mathrm{S}} = \frac{1}{2}\|\mathbf{A} - \sum_{s,t} \mathbf{R}^s \mathbf{D}^{s,t} \mathbf{R}^{t\prime}\|_F^2 \tag{7}$$

$$= \frac{1}{2}\|\mathbf{A} - \mathbf{R}\mathbf{D}\mathbf{R}'\|_F^2, \tag{8}$$

where $\mathbf{R}^t$ is the indicator matrix whose entries are defined in Eqn. 1, $\mathbf{D}^{s,t}$ is the lookup matrix, the entries of the distance matrix $\mathbf{A} \in \mathbf{R}^{N\times N}$ are defined as

$$A_{i,j} \triangleq \|\mathbf{x}_i - \mathbf{x}_j\|_2^2, \tag{9}$$

$\mathbf{R}^{t\prime}$ is the transpose matrix of $\mathbf{R}^t$, $\|\cdot\|_F$ represents the Frobenius norm, and $\mathbf{R}$, $\mathbf{D}$ are defined as

$$\mathbf{R} \triangleq \begin{bmatrix}\mathbf{R}^1 & \cdots & \mathbf{R}^T\end{bmatrix}, \tag{10}$$

$$\mathbf{D} \triangleq \begin{bmatrix}\mathbf{D}^{1,1} & \cdots & \mathbf{D}^{1,T} \\ \vdots & \ddots & \vdots \\ \mathbf{D}^{T,1} & \cdots & \mathbf{D}^{T,T}\end{bmatrix}. \tag{11}$$

This is an unconstrained quadratic program. Setting $\partial\mathcal{F}_{\mathrm{S}}/\partial\mathbf{D}$ as 0, we have

$$\mathbf{EDE} = \mathbf{G}, \tag{12}$$

where

$$\mathbf{E} \triangleq \mathbf{R}'\mathbf{R} = \begin{bmatrix} \mathbf{R}^{1\prime}\mathbf{R}^1 & \cdots & \mathbf{R}^{1\prime}\mathbf{R}^T \\ \vdots & \ddots & \vdots \\ \mathbf{R}^{T\prime}\mathbf{R}^1 & \cdots & \mathbf{R}^{T\prime}\mathbf{R}^T \end{bmatrix}, \tag{13}$$

$$\mathbf{G} \triangleq \mathbf{R}'\mathbf{A}\mathbf{R} = \begin{bmatrix} \mathbf{R}^{1\prime}\mathbf{A}\mathbf{R}^1 & \cdots & \mathbf{R}^{1\prime}\mathbf{A}\mathbf{R}^T \\ \vdots & \ddots & \vdots \\ \mathbf{R}^{T\prime}\mathbf{A}\mathbf{R}^1 & \cdots & \mathbf{R}^{T\prime}\mathbf{A}\mathbf{R}^T \end{bmatrix}. \tag{14}$$

Both $\mathbf{E}$ and $\mathbf{G}$ are computed from the database points and their binary codes. Then, we can solve $\mathbf{D}$ in a closed form by matrix (pseudo-)inversion: $\mathbf{D} = \mathbf{E}^{\dagger}\mathbf{G}\mathbf{E}^{\dagger}$. The *Optimized Symmetric Distance* (OSD) algorithm is decipted in Alg. 1. Below are the details of computing $\mathbf{E}$ and $\mathbf{G}$.

**Compute E**. Let $\mathbf{E}^{s,t} \triangleq \mathbf{R}^{s\prime}\mathbf{R}^t$. We compute the entry on the $m$-th row and $n$-th column of $\mathbf{E}^{s,t}$ as

$$E^{s,t}_{m,n} = \sum_i r^s_{i,m} r^t_{i,n}. \tag{15}$$

Recalling that $r^s_{i,m}$ in Eqn. 1 indicates whether the $i$-th point falls into the $m$-th bucket on the $s$-th partition, we have the observation that $E^{s,t}_{m,n}$ is equal to the number of points which fall into both the $m$-th bucket on the $s$-th partition and the $n$-th bucket on the $t$-th partition.

Computing all the entries $\{E^{s,t}_{m,n}\}$ only need to scan all the binary codes one time. Each $E^{s,t}_{m,n}$ is initialized by 0. For the $i$-th point, $E^{s,t}_{m,n}$ increases by 1 if both $r^s_{i,m}$ and $r^t_{i,n}$ are non-zero. The overall complexity of computing $\{E^{s,t}_{m,n}\}$ is $O(T^2N)$ and the storage complexity is $O(M^2T^2)$.

**Compute G**. According to the definition of the distance matrix $\mathbf{A}$ in Eqn. 9, we compute the element of $\mathbf{G}^{s,t} \triangleq \mathbf{R}^{s\prime}\mathbf{A}\mathbf{R}^t$ in Eqn. 14 as

$$G^{s,t}_{m,n} = \sum_{i,j} r^s_{i,m} A_{i,j} r^t_{j,n} \tag{16}$$

$$= \sum_{i,j} r^s_{i,m} \|\mathbf{x}_i - \mathbf{x}_j\|_2^2 r^t_{j,n} \tag{17}$$

$$= E^{s,s}_{m,m} E^{t,t}_{n,n} (\|\mathbf{c}^s_m - \mathbf{c}^t_n\|_2^2 + \epsilon^s_m + \epsilon^t_n), \tag{18}$$

where

$$\mathbf{c}^s_m \triangleq \begin{cases} \frac{1}{E^{s,s}_{m,m}} \sum_i r^s_{i,m}\mathbf{x}_i, & E^{s,s}_{m,m} > 0 \\ \mathbf{0}, & E^{s,s}_{m,m} = 0, \end{cases} \tag{19}$$

$$\epsilon^s_m \triangleq \begin{cases} \frac{1}{E^{s,s}_{m,m}} \sum_i r^s_{i,m}\|\mathbf{x}_i - \mathbf{c}^s_m\|_2^2, & E^{s,s}_{m,m} > 0 \\ 0, & E^{s,s}_{m,m} = 0, \end{cases} \tag{20}$$

and $\mathbf{c}^t_n$ and $\epsilon^t_n$ are similarly defined. The transformation from Eqn. 17 to Eqn. 18 can be justfied by simply replacing $\|\mathbf{x}_i - \mathbf{x}_j\|_2^2$ with $\|(\mathbf{x}_i - \mathbf{c}^s_m) + (\mathbf{c}^s_m - \mathbf{c}^t_n) + (\mathbf{c}^t_n - \mathbf{x}_j)\|_2^2$.

From Eqn. 19, one can observe that $\mathbf{c}^s_m$ is the center of all the points which fall into the $m$-th bucket on the $s$-th partition, while $\epsilon^s_m$ from Eqn. 20 is the average distortion error between the points and the center.

The computation of $\{G^{s,t}_{m,n}\}$ consists of three steps: 1) scanning all the points to obtain the centers $\{\mathbf{c}^s_m\}$ by Eqn. 19, whose complexity is $O(NPT)$; 2) scanning all the points to obtain the distortion errors $\{\epsilon^s_m\}$ in Eqn. 20 whose complexity is $O(NPT)$; 3) and then computing $\{G^{s,t}_{m,n}\}$ by Eqn. 18 whose complexity is $O(M^2PT^2)$.

**Algorithm 1** Optimized Symmetric Distance.

```
Input: Dataset $\{\mathbf{x}_i\}$; Binary codes $\{\mathbf{b}_i\}$; $T$; $\mathbf{b}_q$
Output: The lookup table $\{d^t_n\}$
1: Compute $\mathbf{E}$ by Eqn. 13 and 15
2: Compute $\{\mathbf{c}^t_m\}$ by Eqn. 19
3: Compute $\{\epsilon^t_m\}$ by Eqn. 20
4: Compute $\mathbf{G}$ by Eqn. 14 and 18
5: Solve $\mathbf{D}$ by Eqn. 12
```

## 5. OPTIMIZED ASYMMETRIC DISTANCE

The goal of optimized asymmetric distance (OAD) is to learn distance functions $\{d^t_n(\mathbf{q})\}$, which map the real-valued query $\mathbf{q}$ to the lookup tables storing the distances between the query and the binary codes for all the $T$ partitions. We choose to learn distance functions instead of generating a complete look matrix offline as done optimized symmetric distance, because the query in OAD is any real-valued point and the number of possible queries is infinite while the query in OSD is encoded to binary codes and hence the number is finite.

### 5.1 Formulation

The objective function for OAD is formulated as follows,

$$\min_{\{d^t_n(\cdot)\}} \mathcal{F}_{\mathrm{A}} \triangleq \frac{1}{2}\mathrm{E}_q\left(\sum_i (\|\mathbf{q} - \mathbf{x}_i\|_2^2 - \sum_{t,n} r^t_{i,n} d^t_n(\mathbf{q}))^2\right), \tag{21}$$

which minimizes the expectation of the misalignment between the original distance and the approximate distance.

It seems that the optimization is not feasible as there is an expectation with respect to the query $\mathbf{q}$ and there is no prior information for the distribution of the query. We will show that the parameters of the distance functions $\{d^t_n(\mathbf{q})\}$ are independent to the query $\mathbf{q}$. On the other hand, the original distance between the query and each database point is needed in the objective function of Eqn. 21. We will show that the functions $\{d^t_n(\cdot)\}$ are learnt only using $\{\mathbf{c}^t_n\}$ (as defined in Eqn. 19), $\{\epsilon^t_n\}$ (as defined in Eqn. 20), and $\mathbf{E}$ (as defined in Eqn. 13). Consequently, the dataset $\{\mathbf{x}_i\}$ can be discarded and is not required in the online computation.

### 5.2 Optimization

The matrix form of Eqn. 21 is written as

$$\mathcal{F}_{\mathrm{A}} = \frac{1}{2}\mathrm{E}_q(\|\mathbf{a}(\mathbf{q}) - \sum_t \mathbf{R}^t\mathbf{d}^t(\mathbf{q})\|_2^2) \tag{22}$$

$$= \frac{1}{2}\mathrm{E}_q(\|\mathbf{a}(\mathbf{q}) - \mathbf{R}\mathbf{d}(\mathbf{q})\|_2^2), \tag{23}$$

where

$$\mathbf{d}(\mathbf{q}) \triangleq \left[\mathbf{d}^1(\mathbf{q})' \quad \cdots \quad \mathbf{d}^T(\mathbf{q})'\right]', \tag{24}$$

$$\mathbf{d}^t(\mathbf{q}) \triangleq \left[d^t_1(\mathbf{q}) \quad \cdots \quad d^t_M(\mathbf{q})\right]', \tag{25}$$

$$\mathbf{a}(\mathbf{q}) \triangleq \left[\|\mathbf{q} - \mathbf{x}_1\|_2^2 \quad \cdots \quad \|\mathbf{q} - \mathbf{x}_N\|_2^2\right]', \tag{26}$$

and $\mathbf{R}^t$ and $\mathbf{R}$ are defined in Eqn. 1 and Eqn. 10, respectively.

**Algorithm 2** Optimized Asymmetric Distance.

```
Input: Dataset $\{\mathbf{x}_i\}$; Binary codes $\{\mathbf{b}_i\}$; $T$; $\mathbf{q}$
Output: The lookup table $\{d_n^t\}$
1: Compute $\mathbf{E}$ by Eqn. 13 and 15
2: Compute $\{\mathbf{c}_n^t\}$ by Eqn. 19
3: Compute $\{\epsilon_n^t\}$ by Eqn. 20
4: Compute inv($\mathbf{E}$)
```

Let the derivative of $\mathcal{F}_\text{A}$ w.r.t. $\mathbf{d}(\mathbf{q})$ as 0. We have

$$\mathbf{d}(\mathbf{q}) = \text{inv}(\mathbf{E})\mathbf{g}(\mathbf{q}), \tag{27}$$

where

$$\mathbf{g}(\mathbf{q}) \triangleq \mathbf{R}'\mathbf{a}(\mathbf{q}) = \begin{bmatrix}\mathbf{a}(\mathbf{q})'\mathbf{R}^1 & \cdots & \mathbf{a}(\mathbf{q})'\mathbf{R}^T\end{bmatrix}', \tag{28}$$

and $\mathbf{E}$ is defined in Eqn. 13, and inv($\mathbf{E}$) represents the (pseudo-) inversion of $\mathbf{E}$.

Let $\mathbf{g}^t(\mathbf{q}) \triangleq \mathbf{R}^{t\prime}\mathbf{a}(\mathbf{q})$. According to the definition of the indicator notation $\mathbf{R}^t$ in Eqn. 1, we compute the $n$-th entry of $\mathbf{g}^t(\mathbf{q})$ as

$$g_n^t(\mathbf{q}) = \sum_i r_{i,n}^t \|\mathbf{q} - \mathbf{x}_i\|_2^2 \tag{29}$$

$$= E_{n,n}^{t,t}(\|\mathbf{q} - \mathbf{c}_n^t\|_2^2 + \epsilon_n^t). \tag{30}$$

where $\mathbf{c}_n^t$, $\epsilon_n^t$ and $E_{n,n}^{t,t}$ are defined in Eqn. 19, Eqn. 20, and Eqn. 15, respectively. The above equation means that only a small number of distances rather than the distances between the query $\mathbf{q}$ and all the database points, are required, and thus the computation cost is reduced. To reduce the online computations, we precompute inv($\mathbf{E}$), $\mathbf{c}_n^t$ and $\epsilon_n^t$. Thus, the online process only computes $\{g_n^t\}$ by Eqn. 30 with $O(PT2^{Q/T})$ time and $\mathbf{d}$ by Eqn. 27 with $O(T^2 2^{2Q/T})$ time, both of which are independent to $N$.

The algorithm of *Optimized Asymmetric Distance* is summarized in Alg. 2.

## 6. DISCUSSIONS

### 6.1 Number of Partitions $T$

#### 6.1.1 Effectiveness vs $T$

The ranking result is often better if the misalignment of the approximate distance towards the original distance is smaller. Thus, we use the mean squared differences between the true distance and the approximate distance to describe the effectiveness, i.e., the objective function under optimal solutions in Eqn. 6 and Eqn. 21 for OSD and OAD, respectively.

The numbers of the optimization variables in Eqn. 6 and Eqn. 21 are $T^2 \times 2^{2Q/T}$ and $T \times 2^{Q/T}$, respectively, both of which are monotonically decreasing with $T$ when $T \leq Q/2$. The following theorem shows how the number of partitions $T$ influences the effectiveness.

THEOREM 1. *Given a positive integer c, and the numbers of partitions $T_1$ and $T_2$ satisfying that $T_1 = cT_2$ and the code length $Q$ is divisible by $T_1$, under the optimal solutions, we have*

$$\mathcal{F}_{S,T_1}^* \geq \mathcal{F}_{S,T_2}^* \tag{31}$$

$$\mathcal{F}_{A,T_1}^* \geq \mathcal{F}_{A,T_2}^* \tag{32}$$

*where $\mathcal{F}_{S,T_i}^*, i = \{1, 2\}$ denotes the optimal objective function value in Eqn. 6 with $T_i$ partitions, and $\mathcal{F}_{A,T_i}^*$ is for Eqn. 21.*

The theorem can be easily proved based on the basic fact that the optimal solution for the case with $T_1$ partitions is a feasible solution for the case with $T_2$ partitions. We omit the detailed proof. Intuitively, this theorem shows that a smaller $T_2$ can lead to a potentially smaller distance approximate error than $T_1$.

#### 6.1.2 Complexity vs $T$

**Storage**: To compute the lookup tables, OSD requires the lookup matrix $\{d_{m,n}^{s,t}\}$ with $O(T^2 2^{2Q/T})$ storage. OAD requires the auxiliary matrix inv($\mathbf{E}$) with $O(T^2 2^{2Q/T})$ storage, $\{\mathbf{c}_m^t\}$ with $O(P2^{Q/T}T)$ storage, and $\{\epsilon_m^t\}$ with $O(T2^{Q/T})$ storage. The storage cost is consistent with the data size, and is generally decreasing with a larger $T$.
**Training time**: For OSD in Alg. 1, it requires $O(T^2 N)$ to compute $\mathbf{E}$, $O(PNT)$ to compute $\{\mathbf{c}_m^s\}$ and $\{\epsilon_m^s\}$, $O(2^{2Q/T}P\, T^2)$ to compute $\{G_{m,n}^{s,t}\}$, and $O(2^{2Q/T}T^2)$ to solve the linear equation of Eqn. 12 to get $\mathbf{D}$. For OAD in Alg. 2, it requires $O(T^2 N)$ to compute $\mathbf{E}$, $O(PNT)$ to compute $\{\mathbf{c}_m^s\}$ and $\{\epsilon_m^s\}$, and $O(2^{3Q/T}T^3)$ to compute the matrix (pseudo-)inversion of $\mathbf{E}$. Practically, to compute $\mathbf{E}$, $\{\mathbf{c}_m^s\}$, $\{\epsilon_m^s\}$, and $\{G_{m,n}^{s,t}\}$ is quite fast. The main time cost is to solve Eqn. 12 for OSD and the matrix (pseudo-)inversion of $\mathbf{E}$ for OAD. Thus, the training time generally decreases with a larger $T$.
**Online query time**: For the online search process, each distance computation of both OSD and OAD requires $O(T)$ time by Eqn. 2. To obtain the lookup tables, OSD requires $O(2^{Q/T}T(T-1))$ preprocessing time by Eqn. 3, while OAD requires $O(PT2^{Q/T})$ and $O(T^2 2^{2Q/T})$ for Eqn. 30 and Eqn. 27, respectively.

### 6.2 Connections with Related Work

In OSD, we first compute the lookup matrix $\{d_{m,n}^{s,t}\}$ and then the lookup table $\{d_n^t\}$ for the query code. In OAD, we compute the functions $\{d_n^t(\cdot)\}$ from the training data and compute the lookup tables for the real-valued query. In this subsection, we show that several existing approaches are special cases of our proposed distances.

The Hamming distance and the symmetric weighted Hamming distance used in [28, 15] is a special case of our proposed symmetric distance under the conditions that $T = Q$ and

$$d_{m,n}^{s,t} = \begin{cases} w_s(m \oplus n), & s = t \\ 0, & s \neq t, \end{cases} \tag{33}$$

where $m \in \{0, 1\}$, $n \in \{0, 1\}$, and $\oplus$ denotes the XOR operation. The coefficient $w_s$ is 1 for Hamming distance or a learnt variable in [28, 15].

In [31], the weights of the weighted Hamming distance are dependent on the un-encoded query as well as the database points. Let $w_s$ be the weight of the $s$th bit. The distance presented in [31] is the special case of our proposed asymmetric distance under the conditions that $T = Q$ and $d_m^s = w_s(b_{q,s} \oplus m)$, where $b_{q,s}$ represents the $s$-th bit of the query code $\mathbf{b}_q$, and $m \in \{0, 1\}$. Similarly, it can be shown that QsRank [33], expectation-based and lower-bound based approaches in [4] are also special cases of our proposed asymmetric distance with different settings for $\{d_m^t(\mathbf{q})\}$.

For product quantization [8], the original $P$-dimensional space is partitioned into multiple disjoint subspaces. Let $T$

be the number of partitions. In each subspace with $P/T$ dimensions, $2^{Q/T}$ clusters are generated and each point is encoded as the concatenation of the index of the nearest cluster center within each subspace. Denote $\hat{\mathbf{c}}_m^s \in \mathbb{R}^{P/T}$ as the $m$-th cluster center in the $s$-th subspace. The symmetric distance introduced in [8] is a special case of our proposed symmetric distance under the conditions that there are $T$ partitions and

$$d_{m,n}^{s,t} = \begin{cases} \|\hat{\mathbf{c}}_m^s - \hat{\mathbf{c}}_n^s\|_2^2 & s = t \\ 0 & s \neq t. \end{cases} \tag{34}$$

We have a similar conclusion for the asymmetric distance introduced in [8] and the proposed optimized asymmetric distance.

## 7. EXPERIMENTS

## 7.1 Experiment Settings

### 7.1.1 High-dimensional datasets

Three widely-used high-dimensional datasets are used, including SIFT1M [8], GIST1M [8], and MNIST [14]. SIFT1M contains 1 million 128-dimensional SIFT descriptors of local image structures, with $10,000$ queries. GIST1M contains 1 million 960-dimensional GIST feature vectors which capture the global image structures, with $1,000$ queries. MNIST has $60,000$ database images, with $10,000$ query images. Each image contains $28 \times 28$ pixels, and we vectorize each image as a 784-dimensional feature vector.

### 7.1.2 Binary code encoding

We adopt several representative binary encoding approaches to encode the high-dimensional data into binary codes.

- Locality sensitive hashing (LSH) [6]: a typical data-independent linear hashing method.
- Iterative quantization (ITQ) hashing [3]: one of the state-of-the-art data-dependent linear hashing methods.
- Kernel-based supervised hashing (KSH) [16]: a typical data-dependent kernel hashing method. Following [16], we use the Gaussian RBF kernel and randomly sample 300 points to construct the hash functions. The supervised information is obtained on $10,000$ randomly sampled points by the Euclidean distance as [16].
- Product quantization (PQ) [8]: a typical non-parametric method, in which the original space is partitioned into multiple disjoint subspaces. Each subspace is clustered into 256 groups.

We generate the binary codes using the above four encoding schemes. For each encoding scheme on each dataset, we generate the codes with different code lengths ranging from 16 to 128.

### 7.1.3 Binary code ranking

After obtaining the binary codes, we rank all the data points based on the distances between their binary codes and the query. Our optimized distances are compared with the following distances.

**Table 2: The methods used in our experiments. ✓(✗) means the corresponding encoding algorithm and the distance are (not) compatible.**

| | Symmetric | | | Asymmetric | | | |
|---|---|---|---|---|---|---|---|
| | OSD | HM | SD[8] | OAD | QsRank[33] | WhRank[31] | AD[8] |
| LSH | ✓ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ |
| ITQ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✗ |
| KSH | ✓ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ |
| PQ | ✓ | ✗ | ✓ | ✓ | ✗ | ✗ | ✓ |

- Hamming distance: The distance is applied on LSH, ITQ and KSH. PQ is unsuitable because the code is the index of the nearest cluster center and the Hamming distance cannot represent the dissimilarity.
- Weighted Hamming distance (WhRank) [31]: WhRank computes the data-adaptive and query-sensitive weight for each bit. The distance between the query and each point is the addition of the weights where the corresponding bit differs.
- Query-sensitive ranking (QsRank) [33]: It assigns float-precision weights for each bit, and the distance is computed as the summation of at most $Q$ weights ($Q$ is the code length). Due to the limitation of QsRank, it is only applied to ITQ, which is based on PCA.
- PQ distances [8]: Both the symmetric distance (SD) and the asymmetric distance (AD) are introduced in [8] for the codes generated by PQ. SD is the distance between the cluster centers, while AD is the distance between the original query point and the cluster center.

To distinguish different distances, we append a suffix to the name of the binary encoding scheme. Our proposed approaches are denoted as '-OSD' and '-OAD' for optimized symmetric distance and optimized asymmetric distance, respectively. '-HM' is used to denote the Hamming distance. '-SD' and '-AD' are referred particularly to as the symmetric and asymmetric distancess in [8] for the codes generated by PQ. For instance, LSH-OAD is interpreted as the LSH encoding scheme with our proposed optimized asymmetric distance. Table 2 summarizes the names.

### 7.1.4 Evaluation criteria

For each query point, we use linear scan to perform the search, i.e., compare it with every database binary code by different distance measures and evaluate the quality of the ranking results. The overall performances are evaluated using two criteria: mean average precision (mAP) and mean average ratio.

The precision at $C$ is defined as the proportion of the number of the true neighbors within the top-ranked $C$ points. The true neighbors of a query are set as the top-2% nearest neighbors in terms of the Euclidean distance as in the work [30]. Average precision (AP) is collected for each query. By averaging over all the queries, we can arrive at the final mAP.

The mean overall ratio [24] reflects the general quality of all top-ranked neighbors. Let $\mathbf{x}_{\hat{q}_c}$ and $\mathbf{x}_{q_c}$ be the $c$-th nearest point of the query $\mathbf{q}$ measured by the Euclidean distance and the approximate distance, respectively. The overall ratio regarding the top-ranked $C$ points is $\text{Ratio}_C(\mathbf{q}) = \frac{1}{C}\sum_{c=1}^{C} \|\mathbf{q} - \mathbf{x}_{q_c}\|_2 / \|\mathbf{q} - \mathbf{x}_{\hat{q}_c}\|_2$. The mean overall ratio is the mean over all the queries. A lower mean overall ratio means a better performance.

## 7.2 Comparison Results

Our proposed approaches (OSD and OAD) have only one parameter, i.e., the number of partitions $T$. As discussed in Sec. 6.1, $T$ controls the trade-off among effectiveness, storage, and speed. In this experiment, we set $T$ in LSH, ITQ and KSH as 2, 3, 6 and 14 for code length 16, 32, 64 and 128, respectively. The length of each partition could be different and we make them as even as possible. For PQ, we follow the number of partitions in the code generation. This parameter will also be studied experimentally in Sec. 7.3.

Fig. 3 shows the experimental results w.r.t. the mAP on the three datasets. From these results, the following observations can be observed.

- Our proposed asymmetric distance achieves the highest mAP in all the settings. This is because the asymmetric distance uses more accurate representation of the query point and generally can achieve better results than the symmetric distance which uses the query binary code directly. Our OAD minimizes the approximation errors, which is beneficial for the ANN search.
- Among the symmetric distances, our OSD performs best, by learning the distance from the original dataset and minimizing the approximation errors.
- For the binary codes generated by KSH and PQ on GIST1M, our OSD is even better than the asymmetric distance WhRank and that of PQ, respectively. This emphasizes that the optimization of minimizing the approximation errors is very promising to improve the ranking accuracy.
- For the PQ code, proposed approaches improve the second best by around 7 percent on the 960-dimensional GIST1M with 1 million points, and gain a small improvement on the other two datasets. This means that our approach achieves significant improvement over the challenging dataset. The reason of different improvements comes from the various data distributions.
- The asymmetric distances QsRank and WhRank generally perform better than the symmetric Hamming distances, which is also demonstrated in [33] and [31].

The experiment results w.r.t. the mean average ratio are shown in Fig. 4 for code length 32. In terms of this criterion, it can be observed that our proposed OAD mostly achieves the lowest mean average ratio. Meanwhile, our proposed OSD is the best among all the symmetric distances.

## 7.3 Effect of $T$

In this experiment, we study the effect of $T$. The experiments run on a server with the Intel(R) Xeon(R) CPU E5-2690 @2.90GHz. The program is in C++ with a single thread to evaluate the time cost of the online query, while in the offline training process, multiple threads are enabled. The results on SIFT1M with the code length 64 generated by ITQ are reported. Similar results can be found on the other datasets with other code lengths.

The results are shown in Fig. 5. Except the similar observations with those in Sec. 7.2, we can also find that the mAP improves for our OSD and OAD as the number of partitions $T$ decreases, because the accuracy of the distance approximation gains in this case. With a small $T$, the query time of OSD reduces. The reason is that only small numbers of additions are required for each distance computation, and the preprocessing time in Eqn. 3 is quite small. For OAD, the query time decreases first and then increases as $T$ increases. The reason is that a smaller $T$ leads to higher preprocessing cost, while a larger $T$ also causes higher distance computational cost. In general, OAD has higher query cost than OSD since more preprocessing computations are involved in OAD. With a larger $T$, the difference between OAD and OSD becomes minor, because the preprocessing cost is small, and the distance computational cost is identical.

For the compared distances, the query time of WhRank and QsRank is longest. This is because for each distance computation in WhRank and QsRank, at most $Q$ additions are required. For our OSD and OAD, the number of additions is $T$, which is much smaller than $Q$.

As illustrated in Fig. 5(c), the training time for both OSD and OAD decreases as $T$ increases. We find the main training cost is to solve Eqn. 12 for OSD and the matrix (psuedo-)inversion of $\mathbf{E}$ for OAD. Time complexities of both routines are smaller for a larger $T$. These results also verify the analysis in Sec. 6.1.

## 8. CONCLUSION

In this paper, we study how to effectively measure the distances for binary code ranking and propose two optimized distance measures: optimized symmetric distance and optimized asymmetric distance. The key novelty lies in the distance table between the (encoded) query and the database codes are explicitly optimized rather than implicitly optimized or heuristically computed as done in the traditional algorithms. Extensive experimental results demonstrate the superiority of our proposed solutions over existing distances.

## Acknowledgements

This work was partially supported by the National Basic Research Program of China (973 Program) under Grant 2014CB347600, ARC Discovery Project DP130103252, National Natural Science Foundation of China (No. 61371192), and Singapore National Research Foundation under its International Research Centre@Singapore Funding Initiative and administered by the IDM Programme Office.

## 9. REFERENCES

[1] A. Babenko and V. S. Lempitsky. The inverted multi-index. In *CVPR*, 2012.

[2] C. Du and J. Wang. Inner product similarity search using compositional codes. *CoRR*, abs/1406.4966, 2014.

[3] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *CVPR*, 2011.

[4] A. Gordo, F. Perronnin, Y. Gong, and S. Lazebnik. Asymmetric distances for binary embeddings. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2014.

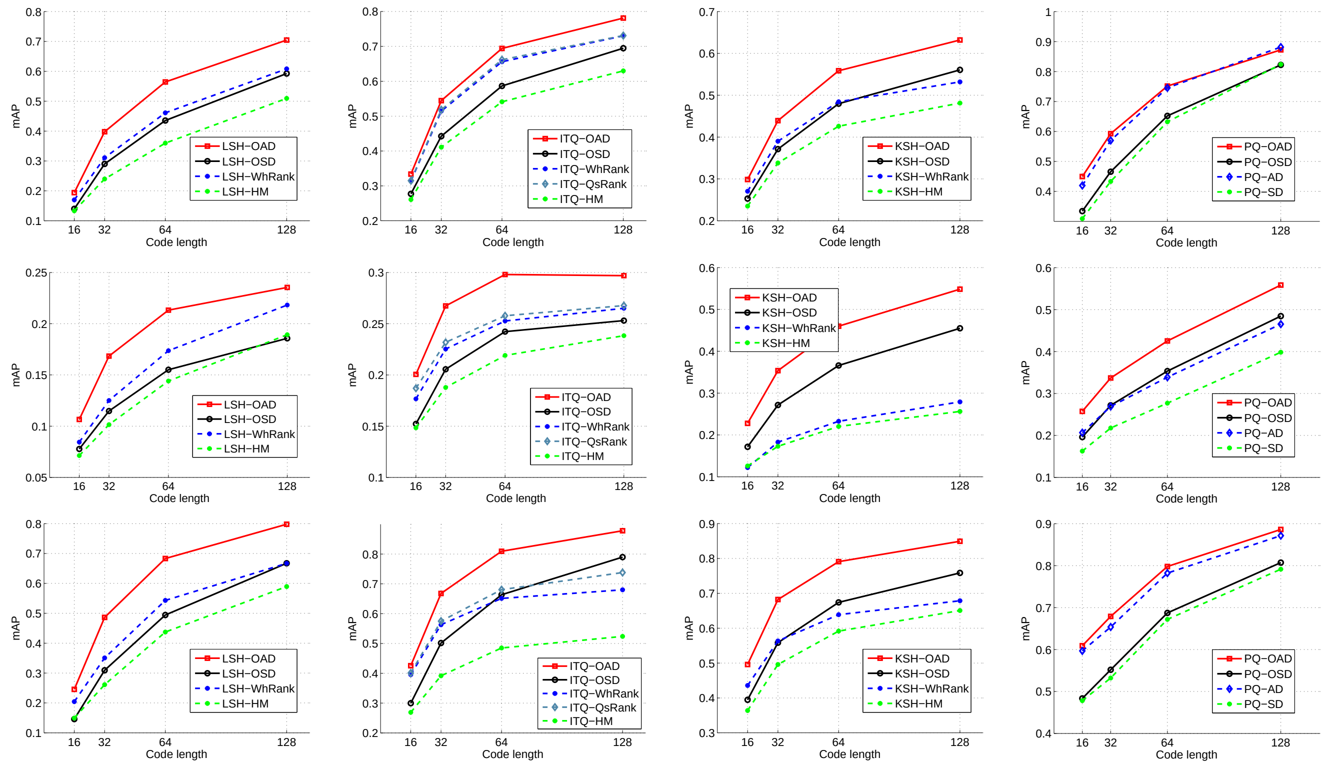


**Figure 3: Results on mAP with with different code lengths. The first, second and third row correspond to SIFT1M, GIST1M, and MNIST respectively. The higher the mAP is, the better the approach is.**

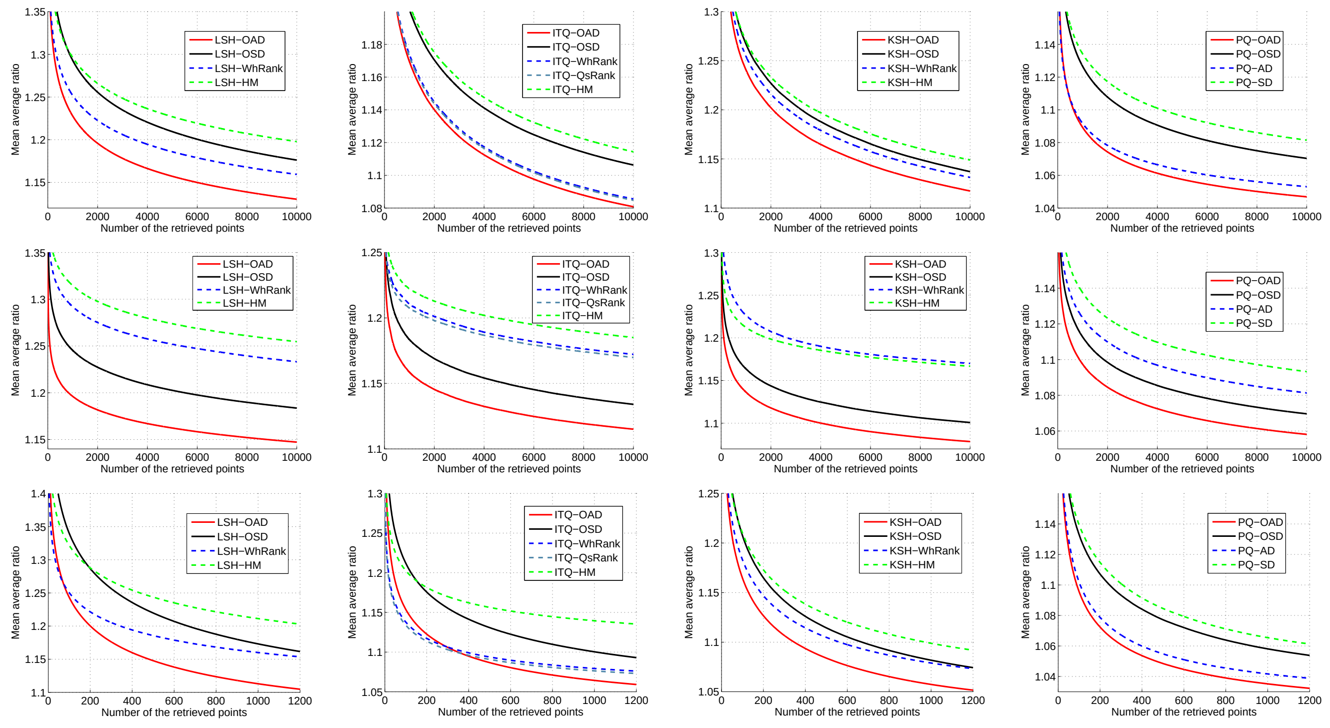
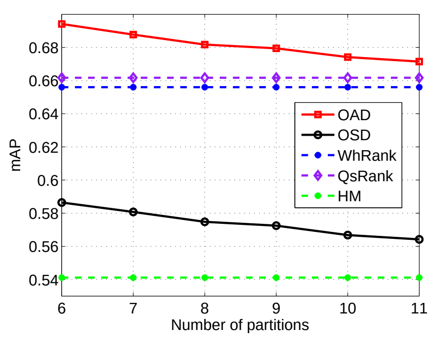
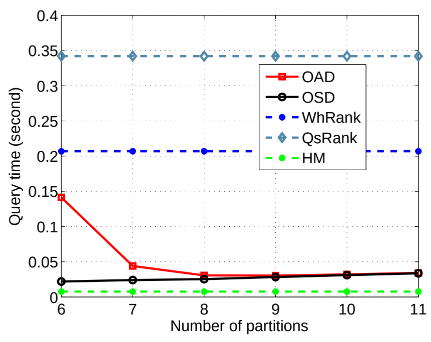


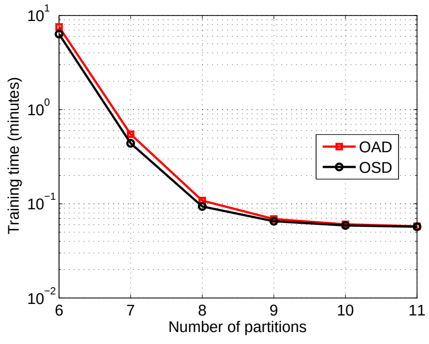
**Figure 4: Results on mean overall ratio with $32$ bits. The first, second and third row corresponds to SIFT1M, GIST1M, and MNIST respectively. The lower the mean average ratio is, the better the approach is.**



(a)



(b)



(c)

**Figure 5: Performance with different numbers of partitions on SIFT1M with 64 bits from ITQ.**

[5] J. Heo, Y. Lee, J. He, S. Chang, and S. Yoon. Spherical hashing. In *CVPR*, 2012.

[6] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, 1998.

[7] M. Jain, H. Jégou, and P. Gros. Asymmetric hamming embedding: taking the best of our bits for large scale image search. In *ACM Multimedia*, 2011.

[8] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(1), 2011.

[9] J. Ji, J. Li, S. Yan, B. Zhang, and Q. Tian. Super-bit locality-sensitive hashing. In *NIPS*, 2012.

[10] W. Kong and W.-J. Li. Isotropic hashing. In *NIPS*, 2012.

[11] W. Kong, W.-J. Li, and M. Guo. Manhattan hashing for large-scale image retrieval. In *SIGIR*, 2012.

[12] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *NIPS*, 2009.

[13] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(6), 2012.

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 1998.

[15] X. Li, G. Lin, C. Shen, A. van den Hengel, and A. R. Dick. Learning hash functions using column generation. In *ICML*, 2013.

[16] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised hashing with kernels. In *CVPR*, 2012.

[17] W. Liu, J. Wang, S. Kumar, and S. Chang. Hashing with graphs. In *ICML*, 2011.

[18] M. Norouzi and D. J. Fleet. Cartesian k-means. In *CVPR*, 2013.

[19] M. Norouzi, A. Punjani, and D. J. Fleet. Fast search in hamming space with multi-index hashing. In *CVPR*, 2012.

[20] M. Raginsky and S. Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. In *NIPS*, 2009.

[21] G. Shakhnarovich, T. Darrell, and P. Indyk. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. The MIT press, 2006.

[22] J. Song, Y. Yang, Z. Huang, H. Shen, and R. Hong. Multiple feature hashing for real-time large scale near-duplicate video retrieval. In *ACM Multimedia*, 2011.

[23] C. Strecha, A. Bronstein, M. Bronstein, and P. Fua. Ldahash: Improved matching with smaller descriptors. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(1), 2012.

[24] Y. Tao, K. Yi, C. Sheng, and P. Kalnis. Efficient and accurate nearest neighbor and closest pair search in high-dimensional space. *ACM Trans. Database Syst.*, 35(3), 2010.

[25] J. Wang, S. Kumar, and S. Chang. Sequential projection learning for hashing with compact codes. In *ICML*, 2010.

[26] J. Wang, J. Wang, J. Song, X.-S. Xu, H. T. Shen, and S. Li. Optimized cartesian $k$-means. *CoRR*, abs/1405.4054, 2014.

[27] J. Wang, J. Wang, N. Yu, and S. Li. Order preserving hashing for approximate nearest neighbor search. In *ACM Multimedia*, 2013.

[28] Y. Weiss, R. Fergus, and A. Torralba. Multidimensional spectral hashing. In *ECCV (5)*, 2012.

[29] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, 2008.

[30] H. Xu, J. Wang, Z. Li, G. Zeng, S. Li, and N. Yu. Complementary hashing for approximate nearest neighbor search. In *ICCV*, 2011.

[31] L. Zhang, Y. Zhang, J. Tang, K. Lu, and Q. Tian. Binary code ranking with weighted hamming distance. In *CVPR*, 2013.

[32] T. Zhang, C. Du, and J. Wang. Composite quantization for approximate nearest neighbor search. In *ICML (2)*, 2014.

[33] X. Zhang, L. Zhang, and H.-Y. Shum. Qsrank: Query-sensitive hash code ranking for efficient $\epsilon$-neighbor search. In *CVPR*, 2012.

[34] X. Zhu, Z. Huang, H. Cheng, J. Cui, and H. T. Shen. Sparse hashing for fast multimedia search. *ACM Trans. Inf. Syst.*, 31(2), 2013.

[35] X. Zhu, Z. Huang, H. T. Shen, and X. Zhao. Linear cross-modal hashing for efficient multimedia search. In *ACM Multimedia*, 2013.