Complementary Binary Quantization for Joint Multiple Indexing

Qiang Fu\(^1\), Xu Han\(^1\), Xianglong Liu\(^*\), Jingkuan Song\(^2\), Cheng Deng\(^3\)

\(^1\) State Key Lab of Software Development Environment, Beihang University, China
\(^2\) Center for Future Media and School of Computer Science and Engineering, University of Electronic Science and Technology of China, China
\(^3\) School of Electronic Engineering, Xidian University, China
fuqiang@nlsde.buaa.edu.cn, steven_han_xu@foxmail.com, xlliu@nlsde.buaa.edu.cn, jingkuan.song@gmail.com, chdeng@mail.xidian.edu.cn

Abstract

Building multiple hash tables has been proven a successful technique for indexing massive databases, which can guarantee a desired level of overall performance. However, existing hash based multi-indexing methods suffer from the heavy redundancy, without strong table complementarity and effective hash code learning. To address the problems, this paper proposes a complementary binary quantization (CBQ) method to jointly learning multiple hash tables. It exploits the power of incomplete binary coding based on prototypes to align the original space and the Hamming space, and further utilizes the nature of multi-indexing search to jointly reduce the quantization loss based on the prototype based hash function. Our alternating optimization adaptively discovers the complementary prototype sets and the corresponding code sets of a varying size in an efficient way, which together robustly approximate the data relations. Our method can be naturally generalized to the product space for long hash codes. Extensive experiments carried out on two popular large-scale tasks including Euclidean and semantic nearest neighbor search demonstrate that the proposed CBQ method enjoys the strong table complementarity and significantly outperforms the state-of-the-arts, with up to 57.76% performance gains relatively.

1 Introduction

Nearest neighbor search plays an important role in many areas like large-scale visual search [Xu et al., 2011; Song et al., 2013; Zhang et al., 2014; Wang et al., 2015], machine learning [Jain et al., 2010; Mu et al., 2014], data mining [Zhang et al., 2016], etc. Nowadays, as the amount of data and information explodes, to solve the problem over gigantic database, the hashing [Wang et al., 2018] based approximate nearest neighbors search technique has been widely studied and successfully applied in practice, owing to its compressed storage and efficient computation.

\(^*\)Corresponding Author

In the literature, Locality-Sensitive Hashing (LSH) was first introduced in [Indyk and Motwani, 1998; Datar et al., 2004] as one of the most essential concepts. It adopted the random projection paradigm to quantize the data into the binary bit (-1 or 1), promising that the nearest neighbors share the similar codes. The binary codes serve as a type of compact feature descriptor, which helps enable fast search. To further improve the discriminative power of the hash codes, later studies have tried to learn the projections by leveraging the information contained in the data [Weiss et al., 2008; Gong and Lazebnik, 2011; Xu et al., 2011; Li et al., 2013; Huang et al., 2013; Jiang and Li, 2014; Shen et al., 2015] and achieve very encouraging progress. However, they can hardly capture the complex inherent structures underlying the data, relying on the linear hash functions. The nonlinear solutions such as deep learning based [Zhu et al., 2016; Lin et al., 2016] and the prototype based [Li et al., 2016] have shown their great power in a wide spectrum of tasks, where usually complex data relations exist.

The binary codes can achieve compressed storage and efficient computations. But in practice for a balanced search performance at a desired level, usually it is required to build a number of hash tables to index the gigantic database [He et al., 2012; Norouzi et al., 2012; Xia et al., 2013; Cheng et al., 2014]. As the prior work has pointed out that when building multiple tables using the binary codes generated by the most of existing hashing methods [Xu et al., 2011; Liu et al., 2015], we inevitably encounter heavy redundancy among the tables, and subsequently often need a huge number of tables, at the cost of significant precision drops and heavy computation. To address the problem, Xu et al. and Liu et al. first studied complementary hashing methods to leverage the mutual benefits between tables. More recently, a boosting strategy named BCH was proposed to ensemble multiple tables that can maximally cover the nearest neighbors with theoretical convergence guarantee [Liu et al., 2017].

Despite the encouraging progress of complementary multi-table indexing, all existing solutions learn the tables in a sequential way, without an overall view of the table relations, and thus still suffer from the table redundancy. Besides, for simplicity they often choose the linear projection based hash functions, which also significantly limit the discriminative power of the hash codes in each table. To address both issues, we develop the prototype based complementary bina-
Figure 1: The geometric view of the binary quantization using ITQ, CH and our CBQ, when building 2 tables with 3 bits per table on a subset of GIST-1M. In each subfigure, the binary codes in different colors respectively correspond to different tables. The average precisions for top 100 retrieval results (AP@100) of the three methods using 3 bits per table are 0.59%, 0.63%, 0.77% respectively.

2 Complementary Binary Quantization

Next, we will formulate our complementary binary quantization (CBQ) method for joint multi-table indexing. Before that we first introduce the notations. Suppose we have a training set with \( n \) training samples \( x_i \in \mathbb{R}^d, i = 1 \ldots n \) of \( d \) dimension, and let denote the total training data by the matrix \( \mathbf{X} = [x_1, x_2, \ldots, x_n] \in \mathbb{R}^{d \times n} \). Our goal is to jointly learn a specific number (i.e., \( L \)) of hash functions \( \{h^{(l)}\}_{l=1}^L \) that can help build informative and complementary hash tables, where each \( h^{(l)}(\cdot) : \mathbb{R}^d \rightarrow \{-1, 1\}^b \) can encode the sample into a \( b \)-length binary codes. Namely, the training data \( \mathbf{X} \) can be encoded as \( \mathbf{Y}^{(l)} = [y_1^{(l)}, y_2^{(l)}, \ldots, y_n^{(l)}] \in \{-1, 1\}^{b \times n} \) by the \( l \)-th hash function.

2.1 Prototype-based Binary Quantization

The prototype technique (e.g., clustering) has been proved very powerful to capture the general metric structure in high dimensional space for large-scale datasets. Therefore, to generate discriminative hash codes for each table, we choose to take advantages of the prototypes to describe the neighbor relations among data. These prototypes will be further assigned unique binary codes for the out-of-sample binary quantization. For multi-table indexing, multiple sets of prototypes will be required which should be able to retrieve the nearest neighbors jointly.

In particular, we learn a set of prototypes \( \mathcal{P}^{(l)} = \{ p_k^{(l)}, p_l^{(l)} \in \mathbb{R}^d \} \) for the \( l \)-th table, and each prototype is associated with a unique \( b \)-bit binary code \( c_k^{(l)} \in \{-1, 1\}^b \), forming a binary codebook \( \mathcal{C}^{(l)} \subseteq \{-1, 1\}^b \).

Based on the prototypes \( \mathcal{P}^{(l)} \) and the corresponding codebooks \( \mathcal{C}^{(l)} \), any new data point \( x \) can be encoded using the code of its nearest prototype in \( \mathcal{P}^{(l)} \), according to specific distance function \( d_o \) (usually Euclidean distance). The prototype-based binary quantization function can be defined as follows:

\[
 h^{(l)}(x) = c_{i(l)(x)}^{(l)}
\]

where \( i(l)(x) = \arg \min_{k} d_o(x, p_k^{(l)}) \).

2.2 Complementary Multi-Table Quantization

With the prototype-based binary quantization, the problem of learning multiple hash tables can be turn to the pursuit of the prototype sets and their codebooks. We aim to formulate the problem and optimize them jointly.

Multi-Table Quantization Loss

When applying multi-table index to facilitate nearest neighbor search, a true search result can be returned as long as it is retrieved successfully by any hash table. Though directly generating long hash codes using existing hashing methods and then dividing them into multiple parts can also help build multiple tables, our empirical observation shows that usually the data partition generated by different parts tends to be correlated, without considering the nature of the multi-index search.

Motivated by this point, for any sample \( x \) we only require at least one of its nearest prototypes in different tables can give the correct search results. Such a nearest prototype can be given by

\[
 (l^*, k^*) = \arg \min_{l} \min_{k} d_o(x, p_k^{(l^*)})
\]

where \( d_o(x, p_k^{(l^*)}) \) is the quantization loss of the sample \( x \) with respect to the prototype \( p_k^{(l^*)} \).

As aforementioned, multi-table indexing attempts to maximally utilize the mutual benefits among tables. Therefore, to guarantee the mutual complementarity, we should minimize the following multi-table quantization loss over all the training data

\[
 \mathcal{L}_{\text{quan}} = \frac{1}{n} \sum_{i=1}^{n} d_o^2(x_i, p_{k_i}^{(l_i^*)})
\]

where \( d_o(x_i, p_{k_i}^{(l_i^*)}) \leq d_o(x_i, p_{l_i}^{(k_i)}), (l, k) \neq (l^*, k^*) \).

Based on this formulation, in order to guarantee the strong complementarity, a straightforward way is to reduce the overlaps among the prototypes of different tables.

Space Alignment Loss

Minimizing the multi-table quantization loss can help us find the most discriminative prototypes. However, how to pursue the prototype-based hash functions, that can preserve the original neighbor relations using binary codes, still remains a critical problem.

Both the prototypes and the binary codes can be treated as a kind of data partition, where the prototype works in the original feature space, while the binary code in Hamming space. Therefore, the desired hash functions, encoding the prototypes into binary codes, actually establish a mapping between the original space and the Hamming space. Specifically, we expect the Hamming distances between the binary codes can
well approximate the original distances between prototypes, i.e., the following fact should hold:
\[
d_{h}(c^{(l)}_{k'}, c^{(l)}_{k}) \approx \lambda d_{o}(p^{(l)}_{k'}, p^{(l)}_{k})
\]
where \(d_{h}(c^{(l)}_{k'}, c^{(l)}_{k})\) is the square root of the Hamming distance between \(c^{(l)}_{k'}\) and \(c^{(l)}_{k}\), and \(\lambda\) is a dynamic scale variable for the two spaces.

Based on this fact, for each sample \(x_{i}\), we expect its distance to any prototype \(p^{(l)}_{k}\) can be approximated by their corresponding Hamming distance. To characterize such distance \(L\) well approximate the original distances between prototypes, minimization which will make both the binary codes and the prototypes well fit the true data distribution.

**The Formulation**

Based on the multi-table quantization loss and the space alignment loss, we can obtain our final formulation for complementary binary quantization:

\[
\min_{\{P^{(l)}\}, \{C^{(l)}\}, \lambda} \mathcal{L} = \mathcal{L}_{\text{quan}} + \mu \mathcal{L}_{\text{align}}
\]
\[
s.t. \quad c^{(l)}_{k} \in \{-1, 1\}^{b}, (c^{(l)}_{k})^{\top} c^{(l)}_{k} \neq b, k' \neq k
\]
\[
d_{o}(x_{i}, p^{(l')}_{k}) \leq d_{o}(x_{i}, p^{(l)}_{k}), (l, k) \neq (l', k')
\]

Here, the first constraint on the binary codes in each \(C^{(l)}\) will guarantee that each prototype will be assigned a unique code. Besides, for each sample \(x_{i}\), the second constraint allows us mainly focus on the capability of its nearest prototype, which is suitable for the multi-indexing search.

3 Joint Table Learning

There are multiple sets of prototypes and codebooks evolved in Problem (6), which are coupled due to the multi-indexing constraints. Moreover, the pursuit of the discrete binary codes further increase the complexity of the problem. Therefore, usually it is quite hard to directly solve it. Fortunately, by transforming the multi-table quantization to a joint one, we can address the above issues using an efficient alternating optimization algorithm and jointly learn multiple tables for a small \(b\) (e.g., \(b \leq 4\)).

3.1 Problem Reformulation

Since our goal is to jointly optimize all prototypes and codebooks of multiple hash tables, we first attempt to reformulate the pursuit of multiple prototype sets to the learning of one joined prototype set.

Since in multi-table search we only focus on the nearest prototype across all tables for each query sample as formulated in Problem (6), this can be done by merging all prototype sets \(P^{(l)}\) into a larger one \(P = \bigsqcup P^{(l)}\) and then finding the nearest prototype from it. Correspondingly, their codebook sets can also be joined as one \(C = \bigcup C^{(l)}\) with repetition. Note that, not that in \(C\) there will be the possibility that the same binary code occurs multiple times, and the total occurrence times for each unique code should be less than \(L\).

Based on this fact, we can construct a one-to-one mapping that converts the original prototype index \((l, k)\) to a uniform one \(m \in \{1, 2, \ldots, M\}\). Subsequently, both the multi-table quantization loss and the space alignment loss can be rewritten as follows:

\[
\min_{\mathcal{P}, \mathcal{C}, \lambda} \mathcal{L} = \min_{\mathcal{P}, \mathcal{C}, \lambda} \sum_{i=1}^{n} d_{o}^{2}(x_{i}, p_{m})
\]
\[
+ \frac{\mu}{nM} \sum_{i=1}^{n} \sum_{m=1}^{M} \|\lambda d_{o}(x_{i}, p_{m}) - d_{h}(c_{m}, c_{m})\|^{2}
\]
\[
s.t. \quad c_{m} \in \{-1, 1\}^{b}, \pi(c_{m}) \leq L
\]
\[
d_{o}(x_{i}, p_{m}) \leq d_{o}(x_{i}, p_{m}), m \neq m_{i}
\]

where \(\pi(c_{m})\) denotes occurrence time for the code \(c_{m}\). \(m_{i}\) is uniform index of prototype which is closest to sample \(x_{i}\).

3.2 Alternating Optimization

With the new formulation, now we can present an efficient alternating optimization algorithm.

**Step 1: Incomplete Coding**

First, suppose we already have the prototypes and the assignment for each sample (to start the algorithm, we initialize them as described in Section 3.3). The problem turns to encoding the prototypes using a subset of binary codes from \(L\) hypercubes (with \(L \times 2^{b}\) candidate codes) that can align the spaces. Such an incomplete coding idea prove to be able to fit the true data distribution using a part of hypercube structure for each table.

Specifically, if supposing in the prototype set \(P\), a subset of prototypes \(p_{1}, \ldots, p_{m}\) have been assigned the binary codes \(c_{1}, \ldots, c_{m}(1 \leq m \leq M)\), then we attempt to find the optimal binary code \(c_{m'}\) for prototype \(p_{m'}\) from the remaining binary codes \(C' = \bigcup_{l \neq l'} \{-1, 1\}^{b} - \{c_{1}, \ldots, c_{m}\}\). With the fact that \(\frac{1}{n} \sum_{i=1}^{n} d_{o}^{2}(x_{i}, p_{m'})\) is a constant, the problem in (7) is equivalent to:

\[
\min_{c_{m} \in C_{m}} \sum_{x_{i}, m' = m, m' \neq m} \|\lambda d_{o}(x_{i}, p_{m'}) - d_{h}(c_{m}, c_{m'})\|^{2}
\]
\[
+ \sum_{x_{i}, m' \neq m'} \|\lambda d_{o}(x_{i}, p_{m'}) - d_{h}(c_{m}, c_{m'})\|^{2}
\]
\[
s.t. \quad \pi(c_{m'}) \leq L
\]

Since there are at most \(2^{b}\) candidate codes for each prototype, the incomplete coding can be directly solved by evaluating
the loss for each candidate and selecting the minimal one by
a greedy algorithm. For \( m = 1 \) as the start of the coding, we
can choose any binary code \( c_1 \) for any prototype, owing to
the high symmetry of the hypercube structure.

**Step 2: Prototype Pursuit**

After obtaining the codebook \( C \), next we can update the proto-
type set \( \mathcal{P} \) to further reduce the multi-table quantization loss
and increase the space consistency. Therefore, we have the
following subproblem:

\[
\min_{\mathcal{P}} \frac{1}{n} \sum_{i=1}^{n} d^2_o(x_i, p_m^*)
+ \frac{\mu}{nM} \sum_{i=1}^{n} \sum_{m=1}^{M} \| \lambda d_o(x_i, p_m) - d_h(c_m^*, c_m) \|^2
\tag{9}
\]

\[s.t. \quad d_o(x_i, p_m^*) \leq d_o(x_i, p_m), m \neq m_i^*\]

The above problem is quite similar to the classic k-means
clustering. Motivated by this observation, we can simply op-
timize the subproblem by iteratively updating the prototype
and the assignment for each sample. Specifically, by fixing
the assignment we can derive the update of \( p_m \) as follows:

\[
p_m = \frac{1}{w_m} \sum_{m_i^* = m} x_i
\tag{10}
\]

where \( w_m \) is the number of samples assigned to the prototype
\( p_m \). The optimal assignment for each sample \( x_i \) in multi-
table search can be found easily as we pointed before:

\[
m_i^* = \arg\min_{m=1, \ldots, M} d_o(x_i, p_m).
\tag{11}
\]

Note that the prototype set will shrink, due to that some
uninformative prototypes without any sample assigned will
be abandoned. This will lead to the incomplete coding.

**Step 3: Rescaling**

The learnt prototype and codebook sets respectively construct
a distribution structure in original data space and Hamming s-
pace. Intuitively, by minimizing the space alignment loss, the
two structures can be well matched. To further minimizing the quantization loss, we still require a proper space scaling
that can maintain the distance measurement consistent across the
two spaces. This can be completed by solving the following
subproblem.

\[
\min_{\lambda} \sum_{i=1}^{n} \sum_{m=1}^{M} \| \lambda d_o(x_i, p_m) - d_h(c_m^*, c_m) \|^2
\tag{12}
\]

whose optimal solution can be given by

\[
\lambda = \frac{\sum_{i=1}^{n} \sum_{m=1}^{M} d_h(c_m^*, c_m)}{\sum_{i=1}^{n} \sum_{m=1}^{M} d_o(x_i, p_m)}.
\tag{13}
\]

**Table Assignment**

After we jointly optimize the prototypes \( \mathcal{P} \) and hash code-
book \( C \), to build \( L \) hash table indices we can simply divide
them to \( L \) parts using a random assignment strategy, following
the two principles: (1) there are no identical hash codes in
the same hash table, and (2) the prototype numbers of every hash
table should be balanced.

Since each binary code corresponds to an integer number
varying in \( \{0, 1, \ldots, 2^b - 1\} \), we conduct the assignment ac-
cording to their numerical order. Namely, we allocate the hash codes and the prototypes with the same number to
different tables, which promises that the same hash codes will
not appear in the same table. Besides, we try to keep the num-
ber of the assigned codes in each table evenly distributed, and
thus balance the indexing capability of multiple tables.

Algorithm 1 lists the main optimization steps of our CBQ
method for the joint multi-table learning.

**3.3 Discussions**

**Initialization**

To start the algorithm, we initialize the prototypes \( \mathcal{P} \) and the
assignment \( m_i^* \) for each samples using the classical K-means
algorithm on the training data set. The number of clusters
(or prototypes) is set to \( M = L \times 2^b \) at first. Based on the
initialization, we also estimate the scaling variable \( \lambda \) using
the full binary codes in \( L \) hypercubes of \( b \) dimension, i.e.,
similar to (13),

\[
\lambda = \frac{1}{M} \sum_{c_m \in \mathcal{C}} \sum_{c_{m'} \in \mathcal{C}} d_h(c_m, c_{m'})
= \frac{1}{n} \sum_{i=1}^{n} \sum_{m=1}^{M} d_o(x_i, p_m).
\tag{14}
\]

**Generating long hash codes**

Till now we have discuss how to generate multiple comple-
mentary hash tables, each of which relies on a short hash
codes (usually \( b \leq 4 \)). In practice, often long binary codes
are required in many tasks for a better performance. However,
it is difficult for our CBQ to jointly learns \( L \times 2^b \)
prototypes for a large \( b \) due to the exponential computation.
Fortunately, in our paper we mainly focus on the generic Eu-
clidean distance and thus can adopt the popular product quan-
tization (PQ) technique to generalize CBQ to product space
for long binary codes. Specifically, in order to generate long
hash codes, we can first divide the original space into multiple
subspaces, then learn a small codes using our CBQ method,
and finally concatenate the small ones in different subspaces
to the long ones in a Cartesian product manner.

---

**Algorithm 1** Complementary Binary Quantization (CBQ).

**Input:** Training set \( X \), hash table number \( L \), code length \( b \)
per table.

**Output:** Hash functions \( \{h_l\}_{l=1}^{L} \), for \( X \) using K-means.

1: Initialize prototype set \( \mathcal{P} \) and the assignment index \( m_i^* \)
   for \( X \) using K-means.
2: Initialize the scale variable \( \lambda \) according to (14).
3: \textbf{repeat} \quad \textbf{for} \( m' = 1, 2, \ldots, M \) \textbf{do}
4: \quad Update the code set \( C \) using the local optimal bi-
   nary code \( c_{m'} \) for \( p_{m'} \) by solving (8).
5: \quad Update \( \lambda \) according to (13).
6: \quad \textbf{end for}
7: Update \( \mathcal{P} \) by iteratively solving (10) and (11).
8: \textbf{end for}
9: \textbf{until} convergence
10: Assign \( \mathcal{P} \) to \( L \) hash tables, generating \( \{h_l\}_{l=1}^{L} \).
### Complexity

To learn $L$ hash tables from $n$ training samples of $d$ dimension, our CBQ algorithm will totally spend $O(4^bndL^2)$ time on the incomplete coding (step 1) that greedily finds the locally optimal code for each prototype in each subspace, and at most $4^bndL^2$ time on prototype (step 2) and scalar update (step 3), mainly deriving from the computation of the distances between training samples and prototypes. With the short code length $b \leq 4$ of each subspace, the term $4^bL^2$ can be treated as a constant. Therefore, the total time for training is linear to the size of the training set. At the online search stage, for each query point the hash functions need $O(2^bdL)$ time to compute the nearest prototype and $O(1)$ time for the code assignment. This is almost the same to the fast hashing methods like LSH and ITQ.

## 4 Experiments

In this section we will evaluate the proposed Complementary Binary Quantization (CBQ) in two popular large-scale tasks including Euclidean and semantic nearest neighbor search.

We first compare CBQ to the state-of-the-art well-known unsupervised hashing algorithms, including the projection based ones like Local Sensitive Hashing (LSH)[Indyk and Motwani, 1998], Iterative Quantization (ITQ)[Gong and Lazebnik, 2011], Spectral Hashing (SH)[Weiss et al., 2008] and Anchor Graph Hashing (AGH)[Takebe et al., 2015], and two representative prototype based ones including Spherical Hashing (SPH)[Heo et al., 2012] and Adaptive Binary Quantization (ABQ)[Li et al., 2016]. Besides, since we mainly tackle the multiple table indexing problem from the practical view, which is quite different from the conventional hashing methods, we also compare our CBQ with two state-of-the-art multi-table methods including Complementary Hashing (CH)[Xu et al., 2011] and Boosting Complementary Hashing (BCH)[Liu et al., 2017].

We have to point out that most of the existing hashing research mainly devoted their efforts to pursuing compact hash codes for Hamming distance ranking or single hash table lookup. Therefore, when applying these methods to building $L$ ($L = 1, 4, 8, 16$) tables with $b$-length ($b = 24$) hash codes in each table, we adopt the common strategy that first generates $bL$-length codes totally and then equally divides them to $L$ parts correspondingly forming $L$ hash tables, similar to Multi-Index Hashing [Norouzi et al., 2012]. For those methods using PQ subspace (ABQ and CBQ), we encode the data in each subspace using $3$ bits.

### 4.1 Evaluation Protocols

In the following experiments we respectively build a different number of hash tables using different methods. As we mainly focus on multi-table indexing technique, we adopt two search schemes over the multiple hash tables, namely the Hamming distance ranking and the hash table lookup.

The Hamming distance ranking should reflect the nature of multi-table search that a point in any hash table with a small Hamming distance to the query will be ranked top. Formally, for the query $x_q$ and any database point $x_i$, we can give their distance over multiple tables as follows

$$d(x_q, x_i) = \min_{i=1,\ldots,L} d_h(y_q^{(i)} , y_i^{(i)})$$

Based on such Hamming distance we can rank all database points in the multi-table indexing nature, and evaluate the ranking performance in terms of the average precision.

Hash table lookup works like the inverted indexing, where the points falling within a small Hamming radius $r$ (usually $r \leq 2$ for efficiency) from the query code are retrieved from each table and merged as the final results. We mainly concern the overall performance by reporting F1-measure.

In the experiments, we randomly select 10,000 and 1,000 samples as the training and the testing set respectively. Besides, to suppress the randomness we repeat all experiments 10 times and report the averaged performance.

### 4.2 Results and Discussions

Multi-table indexing serves as a very fundamental technique in the fields of computer vision, information retrieval, etc. Here, we investigate two of its popular applications including feature matching (Euclidean nearest neighbor search) and image retrieval (semantic nearest neighbor search).

#### Euclidean Nearest Neighbor Search

Here we employ the two widely-used datasets SIFT-1M and GIST-1M [Jegou et al., 2011] that respectively consist of one million 128-D SIFT and 960-D GIST-1M features. In this task, the groundtruth is defined according to Euclidean distance, namely for each query the top $5\%$ points with the smallest Euclidean distances are regarded as its nearest neighbors. We set $\mu$ to 100 on SIFT-1M and 0.2 on GIST-1M.

Table 1 lists the average precision of the top 100 with respect to different number of hash tables. From the table, it is easy to see that the multi-table methods significantly outperform conventional hashing methods like ITQ and ABQ when constructing more hash tables. This further validates

### Table 1: The AP @100 (%) and time cost (seconds) of different hashing methods on SIFT-1M and GIST-1M.

<table>
<thead>
<tr>
<th>Method</th>
<th>SIFT-1M</th>
<th>GIST-1M</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$L=1$</td>
<td>$L=4$</td>
</tr>
<tr>
<td>LSH</td>
<td>27.71±0.63</td>
<td>29.40±0.81</td>
</tr>
<tr>
<td>ITQ</td>
<td>41.00±0.53</td>
<td>30.70±0.33</td>
</tr>
<tr>
<td>SH</td>
<td>48.81±0.84</td>
<td>19.64±3.32</td>
</tr>
<tr>
<td>AGH</td>
<td>31.55±1.71</td>
<td>30.01±1.99</td>
</tr>
<tr>
<td>SPH</td>
<td>39.41±0.89</td>
<td>41.49±0.59</td>
</tr>
<tr>
<td>ABQ</td>
<td>51.53±3.10</td>
<td>32.88±0.57</td>
</tr>
<tr>
<td>CH</td>
<td>50.51±0.94</td>
<td>52.28±0.35</td>
</tr>
<tr>
<td>BCH</td>
<td>45.81±0.93</td>
<td>53.30±0.44</td>
</tr>
<tr>
<td>CBQ(OURS)</td>
<td>52.39±0.71</td>
<td>55.95±0.68</td>
</tr>
</tbody>
</table>
Table 2: MAP (%) and time cost (seconds) of different hashing methods on CIFAR-10 and NUS-WIDE.

<table>
<thead>
<tr>
<th>METHOD</th>
<th>CIFAR-10</th>
<th>NUS-WIDE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L=1</td>
<td>L=2</td>
</tr>
<tr>
<td>LSH</td>
<td>17.58 ± 0.76</td>
<td>18.40 ± 0.16</td>
</tr>
<tr>
<td>ITQ</td>
<td>31.01 ± 0.63</td>
<td>37.37 ± 0.84</td>
</tr>
<tr>
<td>BCH</td>
<td>18.22 ± 0.87</td>
<td>14.84 ± 0.91</td>
</tr>
<tr>
<td>AGB</td>
<td>32.23 ± 1.39</td>
<td>31.11 ± 2.51</td>
</tr>
<tr>
<td>SPH</td>
<td>22.64 ± 1.38</td>
<td>21.27 ± 2.02</td>
</tr>
<tr>
<td>ABQ</td>
<td>18.94 ± 2.56</td>
<td>16.92 ± 0.88</td>
</tr>
<tr>
<td>CH</td>
<td>18.48 ± 2.27</td>
<td>22.02 ± 0.82</td>
</tr>
<tr>
<td>BCH</td>
<td>18.52 ± 1.10</td>
<td>19.95 ± 1.06</td>
</tr>
<tr>
<td>CBQ(Ours)</td>
<td>39.66 ± 1.70</td>
<td>39.37 ± 2.89</td>
</tr>
</tbody>
</table>

Figure 2: F1-Measure performance using different number of hash tables on four datasets when the search radius \( r = 2 \).

Table 3: F1-Measure performance (%) and time cost (ms) of PQ (256 coarse clusters) and CBQ with the same code length in each subspace and same lookup radius on SIFT-1M and GIST-1M.

<table>
<thead>
<tr>
<th>METHOD</th>
<th>SIFT-1M</th>
<th>GIST-1M</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F1 MEASURE</td>
<td>TIME COST</td>
</tr>
<tr>
<td>PQ</td>
<td>23.18</td>
<td>1.22</td>
</tr>
<tr>
<td>CBQ</td>
<td>19.52</td>
<td>0.02</td>
</tr>
<tr>
<td></td>
<td>27.08</td>
<td>0.04</td>
</tr>
</tbody>
</table>

our conclusion that most of existing hashing methods, mainly focusing on compact binary codes, are not suitable for the multi-indexing task. Note that LSH also improves the performance slightly, and this is why many practical applications build table indexing using LSH instead of the learning based ones. Compared to LSH and the other state-of-the-art multi-table methods, our CBQ can obtain the best performance in all cases based on the joint quantization, which indicates that the joint quantization can faithfully help strengthen the complementarity between tables. We can get the similar conclusion from Figure 2 (a) and (b), which depict the F1-measure on the two datasets. In Table 1, we also report the training time and find that CBQ enjoys fast training as we analyzed.

Semantic Nearest Neighbor Search
We further evaluate CBQ on semantic nearest neighbor search for the image retrieval task. We choose two widely-used large-scale image datasets: CIFAR-10 and NUS-WIDE. CIFAR-10 contains of 60K 32 × 32 color images of 10 classes and 6K images in each class, and NUS-WIDE consists of over 269K images with 81 semantic tags. Here, the groundtruth for each query is defined as those samples with common semantics (class or tag) as the query. For both datasets, we extract 4096-D convolutional features for each image using the pretrained VGG network. For NUS-WIDE, we further select 21 most frequent tags for the task in the NUS-WIDE dataset. We set \( \mu \) to 10 on CIFAR-10 and 20 on NUS-WIDE.

Table 2 reports the MAP performance using different methods on CIFAR-10 and NUS-WIDE, where we can get the same observation that the multi-table learning methods increase the performance obviously when constructing more hash tables, and while the conventional compact hashing methods usually decrease. Figure 2 (c) and (d) further investigate the F1-measure performance using hash table lookup. In all cases, we can find that our CBQ performs consistently with the best performance among all methods, which proves that our joint quantization solution promises the complementarity for the semantic neighbor relations.

In Table 3, we also compared our CBQ with the inverted indexing based on product quantization. From the table, we can see that when using the same lookup radius, our CBQ using 8 or 16 tables can get the comparable or even better performance. But in practice, CBQ search is much faster than PQ indexing method.

5 Conclusion
In this paper we mainly studied the hash based multi-indexing problem that has been widely used in many large-scale applications, and proposed a complementary binary quantization (CBQ) method that can jointly pursues multiple complementary and informative hash tables. CBQ method simultaneously considers the nature of multi-table search and the alignment between the original and Hamming space, and can preserve the space consistency using multiple tables. Compared to the state-of-the-art multi-table learning methods, it enjoys both fast training and effective coding based on prototypes and the PQ technique. We have comprehensively evaluated our CBQ method on two fundamental tasks including Euclidean and semantic nearest neighbor search, and our results prove that CBQ can significantly outperform the state-of-the-arts with strong table complementarity.

Acknowledgments
This work was supported by the National Natural Science Foundation of China (61690202) and Microsoft Research Asia Collaborative Research Program.
References


