Fast and Accurate Hashing Via Iterative Nearest Neighbors Expansion

Zhongming Jin, Debing Zhang, Yao Hu, Shiding Lin, Deng Cai, *Member, IEEE,* and Xiaofei He, *Senior Member, IEEE*

***Abstract*—Recently, the hashing techniques have been widely applied to approximate the nearest neighbor search problem in many real applications. The basic idea of these approaches is to generate binary codes for data points which can preserve the similarity between any two of them. Given a query, instead of performing a linear scan of the entire data base, the hashing method can perform a linear scan of the points whose hamming distance to the query is not greater than r_h, where r_h is a constant. However, in order to find the true nearest neighbors, both the locating time and the linear scan time are proportional to $O(\sum_{i=0}^{r_h}\binom{c}{i})$ (c is the code length), which increase exponentially as r_h increases. To address this limitation, we propose a novel algorithm named iterative expanding hashing in this paper, which builds an auxiliary index based on an offline constructed nearest neighbor table to avoid large r_h. This auxiliary index can be easily combined with all the traditional hashing methods. Extensive experimental results over various real large-scale datasets demonstrate the superiority of the proposed approach.**

***Index Terms*—Hashing, indexing, KD-tree, nearest neighbor (NN) search.**

I. Introduction

NEAREST neighbor (NN) search is a fundamental problem and has been applied to various tasks [8], [13], [22], [37], [38], [43], [44], [48]–[53]. A lot of tree-based index structures [2], [5]–[7], [24] (e.g., R-tree [2], KD-tree [7]) have been proposed for NN search. Unfortunately, these approaches perform worse than a linear scan approach when the dimensionality of the space is high [4].

Given the intrinsic difficulty of exact NN search, many hashing approaches are proposed for approximate nearest neighbor (ANN) search [1], [10], [14], [15], [20], [23], [26], [35], [45]–[47]. The key idea of these algorithms is to generate binary codes for data points which can preserve the similarity between any two of them. Given a dataset $\mathbf{X} \in \mathbb{R}^{d\times n}$ consisting of n d-dimensional points, a hashing algorithm uses c hash functions to generate a c-bits hamming embedding $\mathbf{Y} \in \mathbb{B}^{c\times n}$. One way of hashing-based NN search is using hamming ranking. It firstly computes the hamming distance between a query and all points in a dataset. Then, a hamming ranking will be conducted according to the hamming distance. In this process, we can use low-level hardware operation (XOR) to efficiently compute the hamming distance of two binary codes. However, one still needs $O(n)$ to find the NN. One common way to accelerate the search process is using a hash index. By assigning each data point into a c-bits hash bucket corresponding to its c-bits binary code, a hash index can be built. Given a query point, one can use the following three stages to perform the search: 1) coding stage: the query point is converted to a c-bits binary code using the c hash functions; 2) locating stage: all the data points in the buckets that fall within a hamming radius r_h of the binary code of the query are returned; and 3) linear scan stage: a linear scan over these points is conducted to return the required neighbors. It is easy to check that the time complexity of these stages is $O(dc+\sum_{i=0}^{r_h}\binom{c}{i}+\frac{n}{2^c}\sum_{i=0}^{r_h}\binom{c}{i})$. However, in order to find the true NN, the existing hashing methods have to use large r_h, which makes the locating time and the linear scan time exponentially increase and intractable for the search process.

Take Fig. 1(a) as an example, a hashing method generates three hyperplanes l_1, l_2, and l_3 to separate the 2-D space into seven parts. The points in the same part have the same binary codes. The query point is the green star, and its NN are represented by red circles. The remaining points are black squares. By using the hash index, one has to use hamming radius 2 for 100% recall. The number of buckets one needs to locate is $\sum_{i=0}^{2}\binom{3}{i} = 7$ and the number of points need to be validated is 16 (these points involve ten false positive points). Actually, we can use an auxiliary index to obtain high recall without using large hamming radius. As Fig. 1(b) shows, one can find three points $a_i (i = 1, 2, 3)$ from the bucket corresponding to the binary code of the query. By expanding them with their NN, one can also obtain 100% recall. It is clear that the number of buckets that one needs to locate is only one and the number of points need to be validated is seven (these points involve only one false positive point). According to the analysis above, comparing with the original hash index, this idea uses lower locating time and lower linear scan time for the same recall.

Manuscript received July 25, 2013; revised November 12, 2013; accepted January 8, 2014. Date of publication February 7, 2014; date of current version October 13, 2014. This work was supported in part by the National Basic Research Program of China (973 Program) under Grant 2012CB316400, in part by the National Natural Science Foundation of China under Grant 61233011, Grant 61125203, Grant 91120302, and Grant 61222207, and in part by the National Program for Special Support of Top-Notch Young Professionals. This paper was recommended by Associate Editor H. Takagi.

Z. Jin, D. Zhang, Y. Hu, D. Cai, and X. He are with the State Key Laboratory of Computer-Aided Design and Computer Graphics, College of Computer Science, Zhejiang University, HangZhou, Zhejiang 310058, China (e-mail: jinzhongming888@gmail.com; debingzhangchina@gmail.com; huyao001@gmail.com; dengcai@cad.zju.edu.cn; xiaofeihe@cad.zju.edu.cn).

S. Lin is with the Baidu, Inc., Beijing 100085, China (e-mail: linshiding@baidu.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCYB.2014.2302018

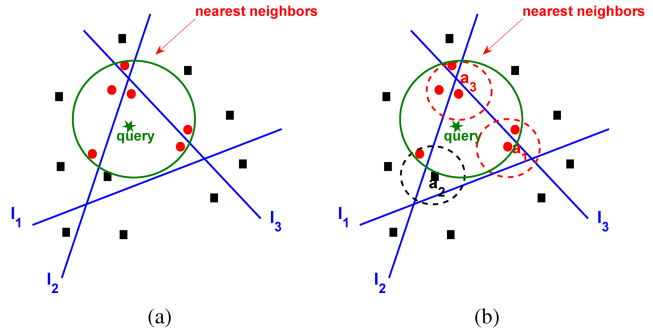


Fig. 1. Illustration for the motivation. A hashing method generates three hyperplanes l_1, l_2, and l_3 to separate the 2-D space into seven parts. The points in the same part have the same binary codes. The query point is the green star and its NN are represented by red circles. The remaining points are black squares. (a) Using the original hash index, one must use hamming radius 2 for 100% recall. (b) By expanding $a_i (i = 1, 2, 3)$ with their NN, one can obtain 100% recall using hamming radius 0.

Based on this idea, we propose a novel algorithm named iterative expanding hashing (IEH), which aims to obtain high recall without using large hamming radius. Specifically, IEH constructs a k nearest neighbors (kNN) table in the offline phase, where k is a parameter. For each data point, its k nearest neighbors are stored in the kNN table. In the online phase, instead of using larger hamming radius to cover more points (as the traditional hashing methods do), IEH uses the kNN table to expand the points, which are obtained from the buckets that fall within a very small hamming radius. By executing an iterative procedure, the performance of IEH can be further improved. The theoretical analysis and experimental results demonstrate that IEH can greatly improve the performance of the traditional hashing methods.

The organization of the paper is as follows. In Section II, we provide a brief description of related work. In Section III, we present the proposed IEH algorithm and analyze its theoretical properties. Then, in Section IV, we compare IEH with traditional hashing methods and KD-tree method on three real world large scale datasets. Finally, in Section V, we provide some concluding remarks.

II. Related Work

A. Hashing-Based ANN Search Methods

The generic hashing problem is summarized as follows. Given n data points $\mathbf{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_n] \in \mathbb{R}^{d \times n}$, one needs to find c hash functions to map a data point $\mathbf{x}$ to a c-bits hash code

$$H(\mathbf{x}) = [h_1(\mathbf{x}), h_2(\mathbf{x}), ..., h_c(\mathbf{x})]$$

where $h_l(\mathbf{x}) \in \{0, 1\}$ is the lth hash function. For the linear projection-based hashing, we have [41]

$$h_l(\mathbf{x}) = sgn(F(\mathbf{w}_l^T \mathbf{x} + t_l)) \quad (1)$$

where $\mathbf{w}_l$ is the projection vector and t_l is the intercept. Different hashing algorithms aim at finding different F, $\mathbf{w}_l$ and t_l with respect to different objective functions.

One of the most popular hashing algorithms is locality sensitive hashing (LSH) [1], which is fundamentally based on the random projection (i.e., LSH uses randomly generated $\mathbf{w}_l$). The F in LSH is an identity function and $t_l = 0$ for mean thresholding.[1] Thus, for LSH, we have

$$h_l(\mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{w}_l^T \mathbf{x} \geq 0 \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

where $\mathbf{w}_l$ is a vector generated from a zero-mean multivariate Gaussian $\mathcal{N}(0, \mathbf{I})$ of the same dimension as the input $\mathbf{x}$. Many extensions for LSH have been proposed [17], [20], [27], [33], [36], [40]. Entropy-based LSH [33] and multiprobe LSH [17], [27] aim to reduce the space requirement in LSH but need much longer time to deal with the query. Kernelized locality sensitive hashing (KLSH) [20] is proposed in the case of high-dimensional kernelized data when the underlying feature embedding is unknown. In addition, the shift-invariant kernels hashing [36], which is based on the random features mapping for shift-invariant kernels, is also proposed. All these methods are fundamentally based on the random projection and ignore the data distribution.

Unlike the above random projection-based hashing methods, a lot of learning-based hashing methods [10], [26], [35], [41], [45] are proposed to make use of the data distribution. Some algorithms [26], [41], [45] exploit the spectral properties of the data affinity (i.e., item-item similarity) matrix for binary coding. The spectral analysis of the data affinity matrix is usually time consuming. To avoid the high computational cost, Weiss *et al.* [45] propose a spectral hashing (SH) method, which makes a strong assumption that data are uniformly distributed. This assumption in SH leads to a simple analytical eigenfunction solution of 1-D Laplacians; however, the geometric structure of the original data is almost ignored, leading to a suboptimal performance. Anchor graph hashing (AGH) [26] is proposed to overcome this shortcoming. It generates anchor points from the data and represents the similarities by these anchor points. In this way, the spectral analysis of the data affinity can be efficiently performed. Some other learning-based hashing methods include iterative quantization (ITQ) [10] which rotates zero-centered data for minimizing the quantization error and spherical hashing (SPH) [35] which learns hypersphere-based hash functions. There are also many efforts on leveraging the label information in hash function learning, which leads to supervised hashing [19], [25], [31] and semisupervised hashing [28], [42].

B. Tree-Based ANN Search Methods

KD-tree method is one of the most popular effective tree-based approaches for ANN search problem. The classical KD-tree algorithm [7] is efficient in low dimensions, but in high dimensions the performance rapidly degrades [4]. Arya *et al.* [3] modify the original KD-tree algorithm for approximate matching, which uses a priority queue to speed up the search in a tree. Beis and Lowe [38] describe a similar KD-tree-based algorithm, but use a stopping criterion based on examining E_{max} (a fixed number) leaf nodes for obtaining better performance. Silpa-Anan and Hartley [39] use multiple

[1] Without loss of generality, we assume that all the data points are centralized to have zero mean.

randomized KD-trees to speed up ANN search. There are some other tree-based methods have been proposed such as R-tree [2], M-tree [5], RP-tree [6], and Ball-tree [24].

C. kNN Graph-Based ANN Search Methods

As described in Section I, the proposed IEH method needs to construct a kNN table in the offline phase. This table has a close connection to the kNN graph. A kNN graph is a directed graph $\mathcal{G} = (\mathbf{X}, \mathbf{E})$ containing a vertex set $\mathbf{X}$ (i.e., data points, $\mathbf{X} \in \mathbb{R}^{d\times n}$) and a directed edge set $\mathbf{E}$. Vertex $\mathbf{x}_i$ is connected to vertex $\mathbf{x}_j$ if $\mathbf{x}_j$ is one of the kNN of $\mathbf{x}_i$.

There are some papers related to applying the kNN graph to ANN search problem [11], [21], [34]. Paredes and Chvez [34] build a kNN graph in the offline phase and use it to minimize the number of distance computations during the nearest-neighbor search. Lifshits and Zhang [21] define a visibility graph and perform nearest-neighbor search by a greedy routing over the graph. Hajebi *et al*. [11] perform hill-climbing starting from a randomly sampled point of the graph. Different from [11], IEH uses hashing method to obtain the initial points, which are closer to the query than randomly sampled points.

III. Iterative Expanding Hashing

A. Notations

Let $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_n]$ be a dataset of size n, and $\mathbf{X} \in \mathbb{R}^{d\times n}$. The kNN of a data point $\mathbf{x} \in \mathbb{R}^d$ in the $\mathbf{X}$ can be denoted by $\mathcal{N}_k(\mathbf{x})$. Given a query $\mathbf{q} \in \mathbb{R}^d$, we wish to find its tNN $\mathcal{N}_t(\mathbf{q})$ in the $\mathbf{X}$. We adopt the Euclidean Distance $\rho(\cdot, \cdot)$ as the distance measure between two points $\mathbf{x}$ and $\mathbf{y}$: $\rho(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2$.

B. Algorithm

The proposed IEH algorithm can be described as follows.

In the offline phase, IEH needs the following preprocesses:

1) use an existing hashing method to generate c hash functions;
2) build the hash index by using the c hash functions;
3) construct the kNN table, which is used for the online iterative expansion.

In the online phase, IEH has three stages.

1) Coding stage: Generate c-bits binary code of the query.
2) Locating stage: The candidates in the buckets that fall within a very small hamming radius are returned.
3) Expansion stage: Iteratively select top p nearest candidates and expand them with their kNN by using the kNN table (the iteration number is s).

We summarize the online procedure of IEH in Algorithm 1.

The main difference between the proposed method and the original hashing technique is that we use expansion stage instead of a large hamming radius. The expansion stage mainly depends on the top nearest candidates and the kNN table. The kNN table is constructed in the offline phase. It stores $\mathcal{N}_k(\mathbf{x})$ for all $\mathbf{x}$ in the $\mathbf{X}$. Since the candidates are close to the query, this expansion gives us some other close candidates and meanwhile avoids large r_h. We also design an iterative procedure, which expands top p nearest candidates in each iteration to implement the query-oriented expansion.

Algorithm 1 Iterative Expanding Hashing (IEH)

Require:

p: the number of top nearest candidates, which are used for expansion in each iteration
k: the number of expansion
s: iteration number
id: temporary id
$\mathcal{M}$: temporary set
Initalization : $id = -1$, $\mathcal{M} = \{\emptyset\}$

coding stage: generate c-bits binary code of the query
locating stage: obtain the candidates from the buckets that fall within a **very small** hamming radius r_h and put them in the $\mathcal{M}$
expansion stage:
for $iter = 1, 2, \cdots, s$ **do**
  select top p nearest candidates from $\mathcal{M}$
  for $i = 1, 2, \cdots, p$ **do**
    for $j = 1, 2, \cdots, k$ **do**
      $id = j$-th NN of i-th top nearest candidate
      if $id \notin \mathcal{M}$ **then**
        $\mathcal{M} = \mathcal{M} \cup \{id\}$
      end if
    end for
  end for
end for
select top t nearest candidates from $\mathcal{M}$ as $\mathcal{N}_t(\mathbf{q})$

C. Dynamic Update of kNN Table

Notice that the kNN table can be dynamically updated. As Fig. 2(a) shows, we wish to insert a new point $\mathbf{x}_m$ into the current kNN table. In order to dynamically update the kNN table, the distance values between the points and their NN should be stored. For example, the point $\mathbf{x}_1$ has NN $\mathbf{x}_5, \mathbf{x}_7, ...$, and the distance value between $\mathbf{x}_1$ and $\mathbf{x}_7$ can be stored as $\rho(\mathbf{x}_1, \mathbf{x}_7)(= \rho(\mathbf{x}_7, \mathbf{x}_1))$. As Fig. 2(b) shows, the process of inserting $\mathbf{x}_m$ into the current kNN table is as follows.

1) Calculate the distance values between $\mathbf{x}_m$ and all points in the current kNN table. For example, $\rho(\mathbf{x}_1, \mathbf{x}_m)$, $\rho(\mathbf{x}_2, \mathbf{x}_m)$, ... can be obtained.
2) Adjust the table according to the distance values. For example, if $\rho(\mathbf{x}_3, \mathbf{x}_9) < \rho(\mathbf{x}_3, \mathbf{x}_m) < \rho(\mathbf{x}_3, \mathbf{x}_2)$, $\mathbf{x}_m$ can be considered as a NN (between $\mathbf{x}_9$ and $\mathbf{x}_2$) of $\mathbf{x}_3$. Since we only maintain kNN of $\mathbf{x}_3$, $\mathbf{x}_7$ will be deleted for $\mathbf{x}_3$. In the implementation, we use linked list to construct the kNN table so as to adjust the table quickly.
3) Insert $\mathbf{x}_m$ and its NN into the table according to the distance values. For example, if $\rho(\mathbf{x}_m, \mathbf{x}_4) < \rho(\mathbf{x}_m, \mathbf{x}_1) < ... < \rho(\mathbf{x}_m, \mathbf{x}_7) < \rho(\mathbf{x}_m, \mathbf{x}_{20})$, the kNN of $\mathbf{x}_m$ are $\mathbf{x}_4$, $\mathbf{x}_1, ..., \mathbf{x}_7, \mathbf{x}_{20}$.

D. Implementation of Online Phase

In the implementation of online phase, we use two key data structures: priority queue and bit map. Firstly, to quickly select top p nearest candidates in each iteration, we use priority queue to store the set $\mathcal{M}$. Secondly, owing to the existence of the duplicate candidates during the expansion stage, we use

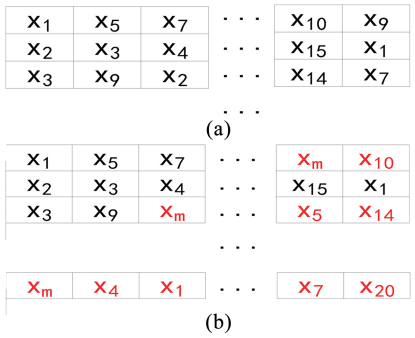


Fig. 2. Illustration for the offline process of constructing the kNN table. It is easy to see that the kNN table can be dynamically updated. (a) Current kNN table. (b) Insert $\mathbf{x}_m$ and its kNN into the current kNN table.

bit map (each bit in the bit map corresponds to a point index) to avoid double counting.

E. Theoretical Analysis

In this section, we provide the theoretical analysis to show that the kNN expansion of IEH based on LSH can guarantee that the exact tNN will be returned with high probability when the parameter k is large enough.

The following well-known lemma which shows the good pairwise similarity preserving property of LSH is adopted.

Lemma 1: (Jonson Lindenstrauss Theorem [16]) Let $U \in \mathbb{R}^{d\times c}$ be a random matrix, with each entry $U_{i,j}$ *i.i.d.* samples from a Gaussian distribution $\mathcal{N}(x|0, 1)$. Let $P_U : \mathbb{R}^d \mapsto \mathbb{R}^c$ be the projection operator that projects a vector in $\mathbb{R}^d$ into the subspace spanned by the column vectors in U. For any two fixed points $\mathbf{x}, \mathbf{q} \in \mathbb{R}^d$ and for a given $\varepsilon \in (0, 1)$, we have $(1-\varepsilon)\|\mathbf{x}-\mathbf{q}\|_2^2 \le \|P_U(\mathbf{x}-\mathbf{q})\|_2^2 \le (1+\varepsilon)\|\mathbf{x}-\mathbf{q}\|_2^2$.

Suppose $\mathbf{X}$ is uniformly distributed. Thus, as Fig. 3 shows, $\mathcal{N}_t(\mathbf{q})$ are randomly distributed within a hypersphere S_{r_1} with radius r_1. After the locating stage, we get p candidates $\mathbf{X}' \in \mathbb{R}^{d\times p}$, $\mathbf{X}' \subset \mathbf{X}$, and the expected distance between a candidate and the query q is $r_2 = 2^{-\frac{1}{d}}(1+\varepsilon)r_1$ based on Lemma 1. For a candidate $\mathbf{x}_i \in \mathbf{X}'$, $\mathcal{N}_k(\mathbf{x}_i)$ are randomly distributed within a hypersphere $S_R^{(i)}$ with radius R. The following theorem shows that under the condition of R, the points obtained by the expansion stage can cover the $\mathcal{N}_t(\mathbf{q})$ with high probability.

Theorem 1: If $R \ge \sqrt{{r_1}^2+{r_2}^2} = \sqrt{4^{-\frac{1}{d}}(1+\varepsilon)^2+1}\cdot r_1$, S_{r_1} can be covered by the union of p $S_R^{(i)}(i = 1, 2..., p)$ with probability at least $1-(\frac{1}{2})^p$.

Proof: In Fig. 3, we can see clearly that if $R \ge \sqrt{{r_1}^2+{r_2}^2}$, each hypersphere centered at a candidate point can cover at least half of the hypersphere S_{r_1}. It is easy to check that in general high-dimensional case, this property also holds true. Based on the randomness of the candidate points, for a given candidate point $\mathbf{x}_i, i = 1, 2, ..., p$, the probability that each point in the hypersphere S_{r_1} is covered by the hypersphere $S_R^{(i)}$ is at least $\frac{1}{2}$. Since there are p independent hyperspheres, S_{r_1} can be covered by the union of p $S_R^{(i)}, i = 1, 2..., p$ with probability at least $1-(\frac{1}{2})^p$. ■

Notice that the lower bound of R is dependent on ε. In IEH, expanding top p NN in each iteration and query-oriented iterative procedure practically decrease this lower bound. The larger the parameter k in the expansion stage, the larger the radius R. Thus, with a large k, IEH can find the exact tNN of the query $\mathbf{q}$ with a very high probability. In addition, experiments in Section IV show that k is not necessary to be very large for IEH to give promising results.

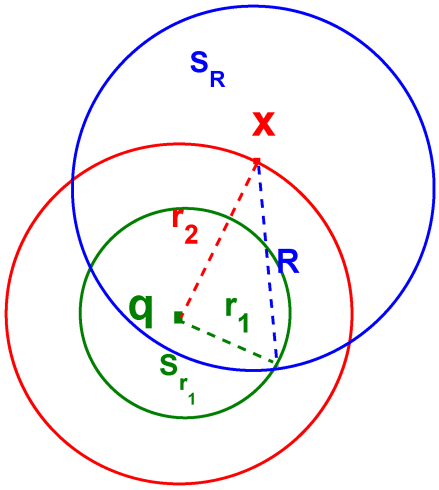


Fig. 3. Illustration for Theorem 1.

TABLE I
STATISTICS OF DATASETS

Data set	# of samples	# of dimensions	Size
CIFAR10	60,000	3072	707M
GIST-1M	1,000,000	384	1.39G
SIFT-1M	1,000,000	128	491M

F. Complexity Analysis

There are three additional costs of IEH.

1) In the offline phase, IEH needs $O(n^2(d+log_2 n))$ to build the kNN table. In the experiment, we show the practical building time is tractable for large scale datasets.
2) In the online phase, the additional space complexity of kNN table is $O(kn)$. Since the practical value of k is small, this additional cost is low.
3) In the online phase, Algorithm 1 shows that instead of using a large hamming radius, IEH only brings a little additional complexity $O(dpks+\sum_{i=1}^{pks} log_2 i)$ for $\mathbf{q}$ in the expansion stage, in which the values of p, k, and s are small. Furthermore, the practical expansion time is very short due to the existence of the duplicate candidates.

IV. EXPERIMENTS

In this section, we evaluate our IEH algorithm on the high-dimensional ANN search problem. Three large-scale real world datasets used in our experiments are (Table I provides some important statistics of the datasets).

1) CIFAR10: It consists of 60 000 images and each image is represented by a 3072-dim vector. This dataset is publicly available[2] and has been used in [10], [25], and [42].

[2]http://www.cs.utoronto.ca/~kriz/cifar.html.

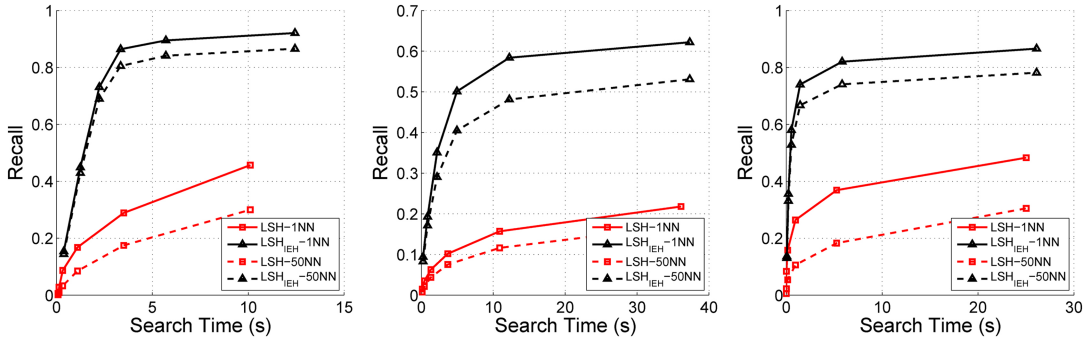


Fig. 4. Recall curves (1000 queries) of LSH and LSH$_{IEH}$. (a) CIFAR10. (b) GIST-1M. (c) SIFT-1M.

TABLE II
ADDITIONAL COSTS OF IEH

Data Set	Additional Offline Time	Additional Online Storage
CIFAR10	156 seconds	12M
GIST-1M	5.56 hours	200M
SIFT-1M	2.5 hours	200M

2) GIST-1M: It contains one million GIST descriptors [32] and each descriptor is represented by a 384-dim vector. This dataset is publicly available[3] and has been used in [25], [35], and citeSSH.
3) SIFT-1M: It contains one million SIFT descriptors [9] and each descriptor is represented by a 128-dim vector. This dataset has been used in [18], [41], and [42].

To reduce the number of candidates obtained by the locating stage, we learn 24-bits, 32-bits and 32-bits hamming embeddings for CIFAR10, GIST-1M and SIFT-1M, respectively. We randomly select 1000 data points as the queries; the remaining can be used to form the gallery database. A returned point is considered to be a true neighbor if it lies in the k closest neighbors (measured by the Euclidean distance in the original space) of the query. For testing the different aspects of performance, we evaluate the algorithms with $k = 1$ and $k = 50$, respectively. There are three parameters in our IEH approach and we empirically set $p = 10$, $k = 50$, and $s = 3$ throughout the experiment. The parameter selection will be discussed in Section IV-E.

All the codes are implemented in C++ and are tested on a machine with Intel Xeon(R) E7450 CPU (24 cores) and 256 GB memory. In the offline phase, we use the multithread technology to speed up the process of constructing the kNN table. On the contrary, for the sake of fairness, we do not use the multithread technology for all the methods in the online phase.

As we discussed in Section III-F, IEH brings some additional costs. Table II shows the additional offline time and additional online storage. We can clearly see that the additional costs are tractable for large-scale datasets. In addition, we only use a single machine to run the offline preprocess in this experiment. For a large-scale dataset, this preprocess can be further speeded up by using multimachines in parallel.

[3]http://horatio.cs.nyu.edu/mit/tiny/data/index.html.

TABLE III
50 NN MEAN AVERAGE PRECISION OF ORIGINAL HASHING METHOD AND IEH

DataSet	Hashing Method	Original	IEH
CIFAR10	LSH	0.014	0.079
	KLSH	0.071	0.073
	SH	0.039	0.108
	AGH	0.037	0.076
	SPH	0.031	0.103
GIST-1M	LSH	0.004	0.013
	KLSH	0.014	0.021
	SH	0.010	0.024
	AGH	0.006	0.016
	SPH	0.009	0.022
SIFT-1M	LSH	0.024	0.054
	KLSH	0.042	0.050
	SH	0.051	0.069
	AGH	0.039	0.060
	SPH	0.036	0.063

A. Comparison With Original Hashing Method

Since our algorithm IEH does not depend on any specific hashing method, different hashing methods may lead to different results. We use five state-of-the-art hashing algorithms.

1) LSH: Locality sensitive hashing [1], which is fundamentally based on the random projection.
2) KLSH: Kernelized locality sensitive hashing [20], which generalizes the LSH method to the kernel space.
3) SH: Spectral hashing [45], which is based on quantizing the values of analytical eigenfunctions computed along PCA directions of the data.
4) AGH: Anchor graph hashing [26], which constructs an anchor graph to speed up the spectral analysis.
5) SPH: SPherical hashing [35], which uses a hypersphere-based hash function to map data points into binary codes.

It is important to note that LSH is a linear method, whereas the remaining four methods are nonlinear methods. For KLSH and AGH, we use the Gaussian kernel and the width parameter σ is estimated by randomly choosing 3000 samples and let σ be equal to the average of the pairwise distances. Both KLSH and AGH need to select m supporting samples, and we use the same m samples (generated by K-means) for them. The parameter m is fixed to be 100 throughout of the experiment.

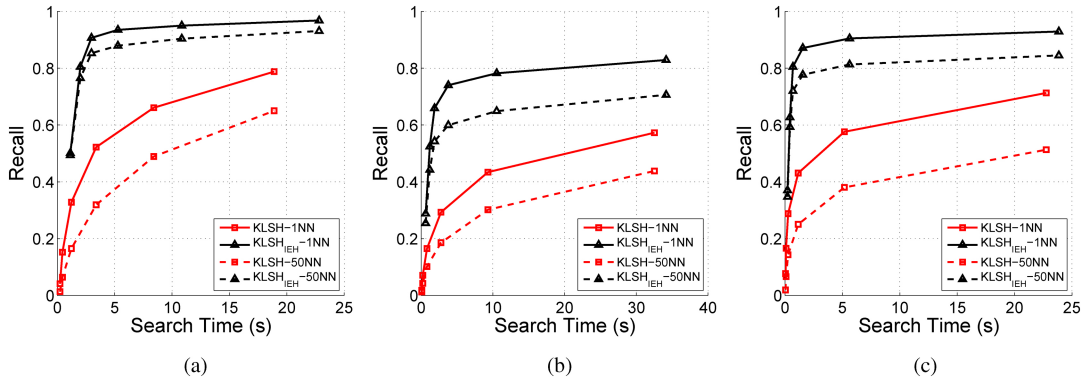


Fig. 5. Recall curves (1000 queries) of KLSH and KLSH$_{IEH}$. (a) CIFAR10. (b) GIST-1M. (c) SIFT-1M.

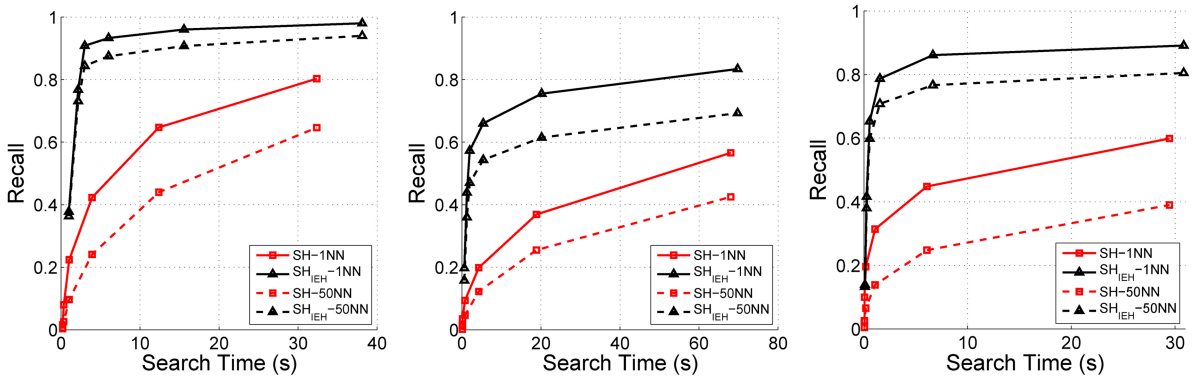


Fig. 6. Recall curves (1000 queries) of SH and SH$_{IEH}$. (a) CIFAR10. (b) GIST-1M. (c) SIFT-1M.

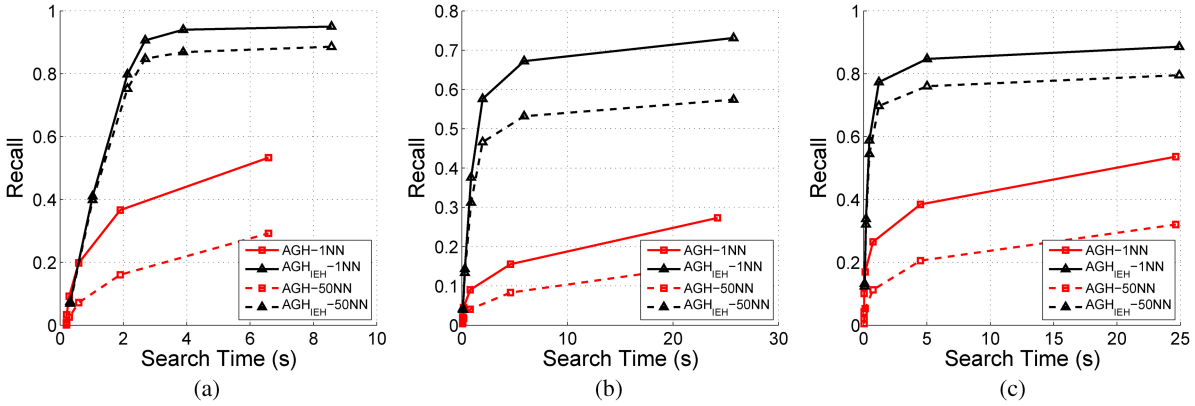


Fig. 7. Recall curves (1000 queries) of AGH and AGH$_{IEH}$. (a) CIFAR10. (b) GIST-1M. (c) SIFT-1M.

Following [12], [25], [26], [35], [41], and [42], we use the recall curve to evaluate the performance of IEH comparing with the original hashing method. In order to consider all the processing time (i.e., coding time, locating time, linear scan time, and expanding time) in the online phase, we use the real search time (in seconds) instead of the number of retrieved samples (the original recall curve used) as x-axis. Each point in the recall curve corresponds to a given hamming radius ranging from zero to five.

Fig. 4 shows 1NN and 50 NN recall curves of LSH with IEH extension compared with the original LSH method. LSH$_{IEH}$-1NN represents the LSH with IEH extension for searching 1NN. Figs. 5–8 show the performance of IEH extension on other four hashing methods.

As we discussed in Section III-F, given a fixed hamming radius, IEH extension spends more time than the original hashing approach simply because IEH needs the iterative kNN expansion stage. However, IEH extension obtains significantly better recall. Spending the same amount of time, IEH can obtain significantly higher recall mainly because the expanded candidates involve a few false positive points. Moreover, it is surprising that IEH has a higher recall and a lower search time simultaneously than the original hashing method in many cases. For example, in CIFAR10, LSH-1NN spends 10.11s

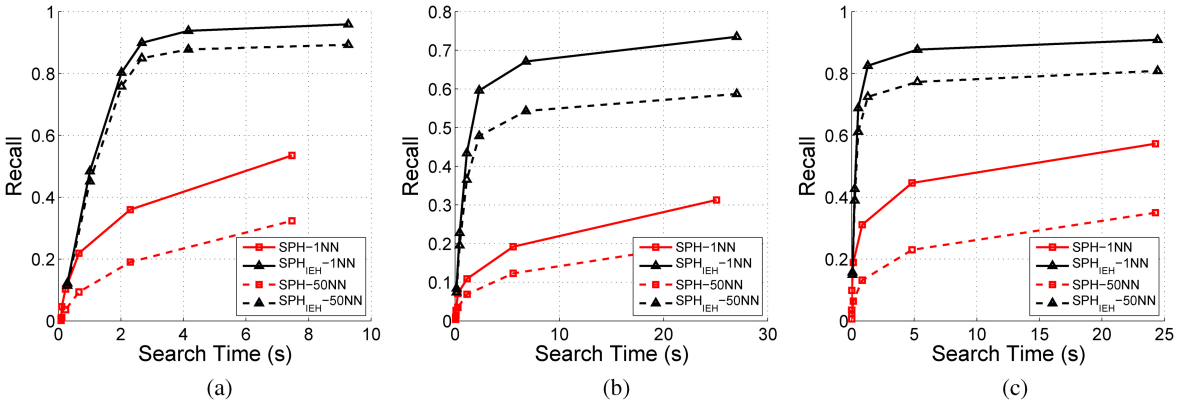


Fig. 8. Recall curves (1000 queries) of SPH and SPH$_{IEH}$. (a) CIFAR10. (b) GIST-1M. (c) SIFT-1M.

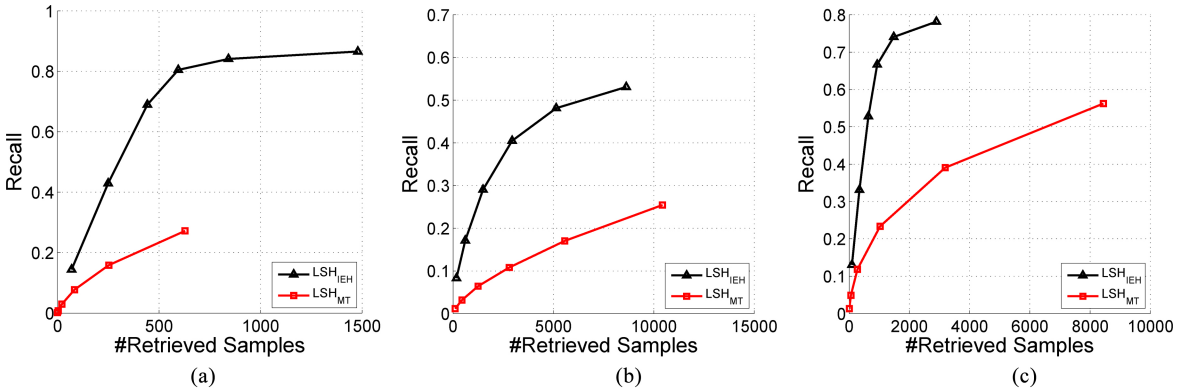


Fig. 9. 50 NN recall curves (recall versus the number of retrieved samples) of LSH$_{MT}$ and LSH$_{IEH}$. (a) CIFAR10. (b) GIST-1M. (c) SIFT-1M.

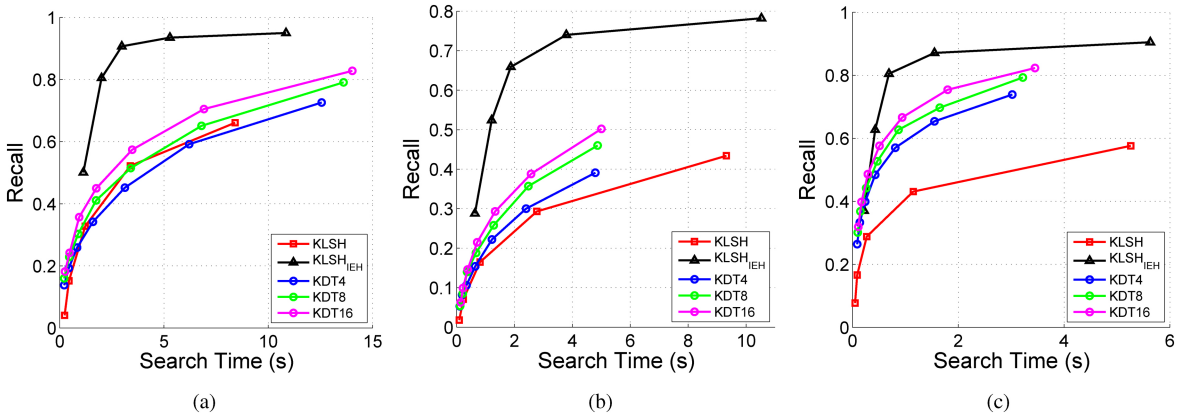


Fig. 10. 1 NN recall curves (1000 queries) of KLSH, KLSH$_{IEH}$ and KD-tree. (a) CIFAR10. (b) GIST-1M. (c) SIFT-1M.

obtaining 45.7% recall and LSH$_{IEH}$-1NN spends only 2.15s obtaining 73.1% recall. These figures show clearly the great superiority of IEH over the original hashing method.

Table III shows the 50 NN mean average precision (mAP) of original hashing method and the proposed expanding method (IEH) on three datasets. We can see clearly that IEH extension consistently achieves higher mAP than original hashing method.

B. Comparison With Multitable LSH

In this section, we compare the IEH with multitable-based hashing method. Fig. 9 and Table IV show the 50 NN recall—number of retrieved samples curves, and mAP of multitable LSH (LSH$_{MT}$), and the proposed expanding method (LSH$_{IEH}$) on three datasets. In these experiments, we build a three-tables LSH. We can clearly see that IEH extension consistently achieves higher recall and mAP than multitable-based hashing method.

TABLE IV
50 NN MEAN AVERAGE PRECISION OF LSH$_{MT}$ AND LSH$_{IEH}$

DataSet	LSH$_{MT}$	LSH$_{IEH}$
CIFAR10	0.023	0.079
GIST-1M	0.005	0.013
SIFT-1M	0.038	0.054

C. Comparison With KD-Tree

In this experiment, we use the recall curve to compare KLSH$_{IEH}$ (IEH algorithm based on KLSH) with KD-tree

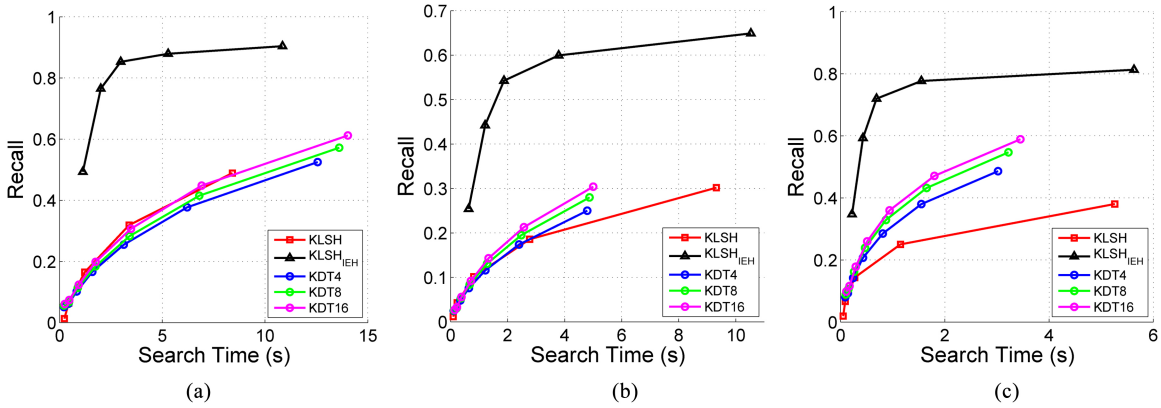


Fig. 11. 50 NN recall curves (1000 queries) of KLSH, KLSH$_{IEH}$ and KD-tree. (a) CIFAR10. (b) GIST-1M. (c) SIFT-1M.

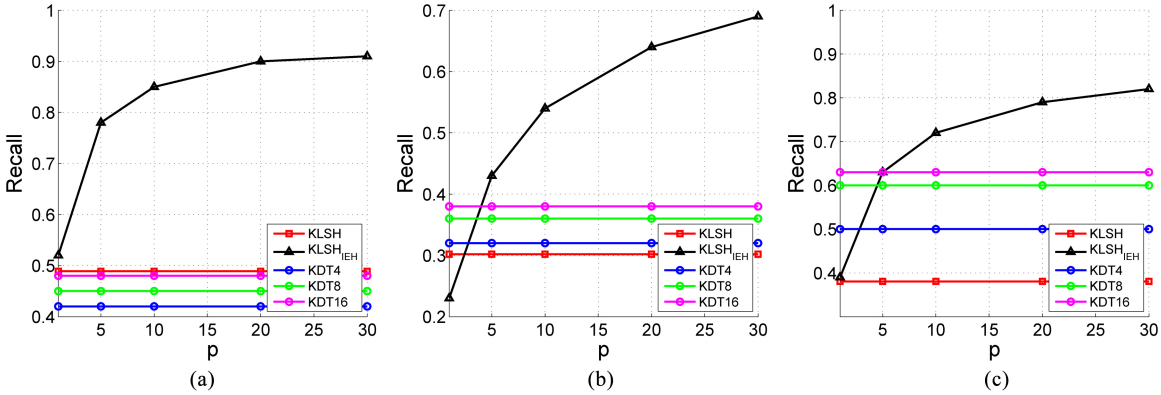


Fig. 12. 50 NN recall (1000 queries) of KLSH$_{IEH}$ versus the parameter p at hamming radius 2. (a) CIFAR10. (b) GIST-1M. (c) SIFT-1M.

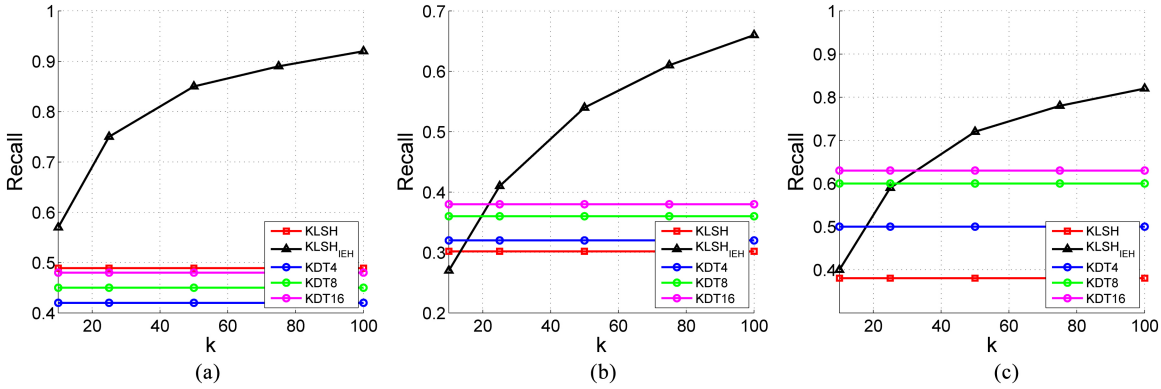


Fig. 13. 50 NN recall (1000 queries) of KLSH$_{IEH}$ versus the parameter k at hamming radius 2. (a) CIFAR10. (b) GIST-1M. (c) SIFT-1M.

TABLE V
STORAGE OF ALL METHODS

Data Set	KLSH	KLSH$_{IEH}$	KDT4	KDT8	KDT16
CIFAR10	10M	22M	14M	28M	57M
GIST-1M	10M	210M	310M	510M	1.01G
SIFT-1M	10M	210M	250M	496M	909M

of Fast Library for ANN (FLANN) [29][4] on the datasets. FLANN is one of the most popular libraries for the ANN search problem. The KD-tree of FLANN is implemented in [38] and [39]. KDT4 represents the KD-tree method using four trees and so as the KDT8 and KDT16. Each point in the recall curve of KD-tree corresponds to a given check number: 32, 64, 128, 256, 512, 1024, and 2048.The check number is a parameter of KD-tree (please see the manual of FLANN [30] for details). For KLSH$_{IEH}$ and KLSH, each point corresponds to a given hamming radius ranged from 0 to 4.

Figs. 10 and 11 show 1NN and 50 NN recall curves of KLSH, KLSH$_{IEH}$, and KD-tree. We can clearly see that IEH has more advantages than KD-tree with the growth of dimensionality of data space. Furthermore, the superiority of IEH is more obvious for searching 50 NN. Both figures demonstrate that IEH significantly outperforms KD-tree on three different datasets.

Table V shows the memory usages of the indices of KLSH, KLSH$_{IEH}$, and KD-tree. It demonstrates that IEH occupies

[4]http://www.cs.ubc.ca/~mariusm/index.php/FLANN

TABLE VI
PROCESSING TIME (1000 QUERIES) OF IEH ON GIST-1M DATASET

Time	Hashing Method	$r_h = 0$	$r_h = 1$	$r_h = 2$	$r_h = 3$	$r_h = 4$	$r_h = 5$
Coding Time (s)	LSH	0.03	0.03	0.03	0.03	0.03	0.03
	KLSH	0.06	0.06	0.06	0.06	0.06	0.06
	SH	0.08	0.08	0.08	0.08	0.08	0.08
	AGH	0.08	0.08	0.08	0.08	0.08	0.08
	SPH	0.03	0.03	0.03	0.03	0.03	0.03
Locating Time (s)	LSH	0.001	0.001	0.08	0.83	4.97	25.43
	KLSH	0.001	0.01	0.12	0.83	4.78	22.14
	SH	0.001	0.001	0.12	1.09	7.24	31.86
	AGH	0.001	0.001	0.05	0.65	4.31	23.58
	SPH	0.001	0.001	0.12	0.7	4.52	23.04
Linear Scan Time (s)	LSH	0.07	0.38	1.21	2.82	5.88	10.63
	KLSH	0.03	0.17	0.63	1.89	4.62	10.31
	SH	0.001	0.09	0.59	3.05	11.55	36.12
	AGH	0.001	0.001	0.01	0.07	0.23	0.58
	SPH	0.001	0.02	0.1	0.4	0.92	1.99
Expanding Time (s)	LSH	0.21	0.51	1.01	1.54	1.53	1.36
	KLSH	0.73	1.22	1.01	1.17	1.05	0.82
	SH	0.63	1.28	1.43	1.28	1.11	0.9
	AGH	0.04	0.25	0.91	1.41	1.38	1.39
	SPH	0.09	0.35	0.89	1.11	0.98	0.85

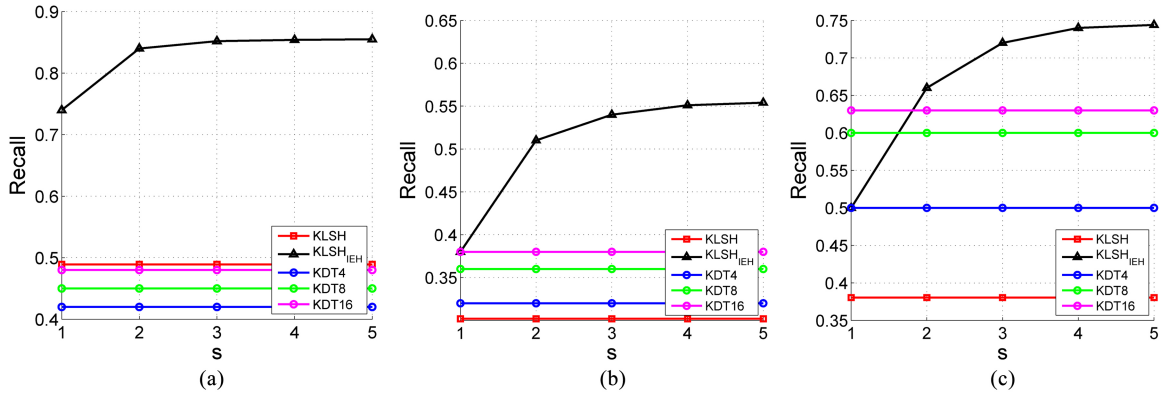


Fig. 14. 50 NN recall (1000 queries) of KLSH$_{IEH}$ versus the parameter s at hamming radius 2. (a) CIFAR10. (b) GIST-1M. (c) SIFT-1M.

TABLE VII
50 NN SEARCH TIME (SECONDS/1000 QUERIES) OF KLSH$_{IEH}$ VERSUS PARAMETER p AT HAMMING RADIUS 2

Data Set	KLSH$_{IEH}$ ($k = 50, s = 3$)					KLSH and KD-Tree
	$p = 1$	$p = 5$	$p = 10$	$p = 20$	$p = 30$	
CIFAR10	1.45	2.15	2.83	3.67	4.45	8.21
GIST-1M	0.91	1.34	1.82	2.65	3.38	9.26
SIFT-1M	0.31	0.48	0.65	0.93	1.19	5.22

TABLE VIII
50 NN SEARCH TIME (SECONDS/1000 QUERIES) OF KLSH$_{IEH}$ VERSUS THE PARAMETER k AT HAMMING RADIUS 2

Data Set	KLSH$_{IEH}$ ($p = 10, s = 3$)					KLSH and KD-Tree
	$k = 10$	$k = 25$	$k = 50$	$k = 75$	$k = 100$	
CIFAR10	1.58	2.09	2.83	3.47	4.11	8.21
GIST-1M	1.01	1.33	1.82	2.26	2.69	9.26
SIFT-1M	0.36	0.48	0.65	0.8	0.95	5.22

a little memory comparing with KD-tree. Considering both memory usage and speed, KLSH$_{IEH}$ outperforms the state-of-the-art KD-tree implementation (FLANN) on a large scale ANN search problem.

D. Time Complexity

As we mentioned in Section III-B, the main difference between the proposed method IEH and original hashing method is the expansion stage. Table VI shows the processing time (1000 queries) of all stages in IEH on GIST-1M dataset. It is clear to see that the expanding time is not significant, especially for a large hamming radius.

E. Parameter Selection

As discussed before, IEH has three essential parameters p, k, and s. In the previous experiments, we empirically set $p = 10$, $k = 50$, and $s = 3$. In this experiment, we try to examine how the performance of IEH can be affected by these parameters.

Figs. 12–14 show the 50 NN recall of KLSH$_{IEH}$ varies as the p, k, and s vary at hamming radius 2, respectively. These figures also show the 50 NN recall of KLSH at hamming radius 4 and the 50 NN recall of KD-tree, which has the same search time as KLSH. Tables VII–IX show that the 50 NN search time (in seconds) of KLSH$_{IEH}$ varies as the

TABLE IX
50 NN SEARCH TIME (SECONDS/1000 QUERIES) OF KLSH$_{IEH}$ VERSUS PARAMETER s AT HAMMING RADIUS 2

Data Set	KLSH$_{IEH}$ ($p = 10, k = 50$)					KLSH and KD-Tree
	$s = 1$	$s = 2$	$s = 3$	$s = 4$	$s = 5$	
CIFAR10	2.32	2.77	2.83	2.85	2.86	8.21
GIST-1M	1.3	1.68	1.82	1.87	1.89	9.26
SIFT-1M	0.45	0.58	0.65	0.68	0.69	5.22

p, k, and s vary at hamming radius 2, respectively. It is no surprise to see that the recall increases as the p, k, or s increases (because we examine more candidate points). With larger p, k, or s, KLSH$_{IEH}$ spends more time searching the NN. This gives the user a way to balance the speed and accuracy based on different applications. We can see clearly that $p = 10, k = 50, s = 3$ are reasonable for considering both high recall and low search time.

V. CONCLUSION

In this paper, we propose a novel algorithm named IEH which aims to obtain high recall without using large hamming radius. IEH uses very small hamming radius and iteratively expands a few nearest candidates with their kNN. In this way, it obtains high recall and low search time simultaneously. The theoretical analysis and experimental results demonstrate that it can greatly improve the search results of traditional hashing methods. Moreover, the experiments also show that our method achieves a better performance than the KD-tree method, which is a state-of-the-art solution for real world nearest-neighbor search problem. In the future, we will use FLANN to accelerate the offline preprocess and build multitables to further improve the online performance. Furthermore, we will consider designing a mechanism that can automatically adjust the parameters (i.e., p, k, and s).

REFERENCES

[1] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *Commun. ACM*, vol. 51, no. 1, pp. 117–122, 2008.

[2] L. Arge, M. Berg, H. Haverkort, and K. Yi, "The priority R-tree: A practically efficient and worst-case optimal R-tree," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, Jun. 2004, pp. 347–358.

[3] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Wu, "An optimal algorithm for approximate nearest neighbor searching in fixed dimensions," *J. ACM*, vol. 45, no. 6, pp. 891–923, 1998.

[4] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is 'nearest neighbor' meaningful?" in *Proc. 7th Int. Conf. Database Theory*, Jan. 1999, pp. 217–235.

[5] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An efficient access method for similarity search in metric spaces," in *Proc. 23rd Int. Conf. Very Large Data Bases*, Aug. 1997, pp. 426–435.

[6] S. Dasgupta and Y. Freund, "Random projection trees and low dimensional manifolds," in *Proc. 40th Annu. ACM Symp. Theory Comput.*, May 2008, pp. 537–546.

[7] J. Freidman, J. Bentley, and R. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Trans. Math. Softw.*, vol. 3, no. 3, pp. 209–226, 1997.

[8] Y. Gao, B. Zheng, G. Chen, Q. Li, and X. Guo, "Continuous visible nearest neighbor query processing in spatial databases," *Very Large Databases J.*, vol. 20, no. 3, pp. 371–396, 2011.

[9] D. G. Lowe, "Distinctive image features from scale invariant keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[10] Y. Gong and S. Lazebnik, "Iterative quantization: A procrustean approach to learning binary codes," in *Proc. IEEE Conf. Comput. Vision Pattern Recogn.*, Jun. 2011, pp. 817–824.

[11] K. Hajebi, Y. Abbasi-Yadkori, H. Shahbazi, and H. Zhang, "Fast approximate nearest-neighbor search with k-nearest neighbor graph," in *Proc. 22nd Int. Joint Conf. Artif. Intell.*, Jul. 2011, pp. 1312–1317.

[12] J. He, R. Radhakrishnan, S. F. Chang, and C. Bauer, "Compact hashing with joint optimization of search accuracy and time," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, Jun. 2011, pp. 753–760.

[13] P. Jain, B. Kulis, and K. Grauman, "Fast similarity search for learned metrics," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 31, no. 12, pp. 2143–2157, Dec. 2009.

[14] Z. Jin, Y. Hu, Y. Lin, D. Zhang, S. Lin, D. Cai, *et al.*, "Complmentary projection hashing," in *Proc. IEEE Int. Conf. Comput. Vision*, Dec. 2013, pp. 257–264.

[15] Z. Jin, C. Li, Y. Lin, and D. Cai, "Density sensitive hashing," *IEEE Trans. Cybern.*, vol. PP, no. 99, Oct. 2013.

[16] W. Johnson and J. Lindenstrauss, "Extensions of Lipschitz mappings into a hilbert space," *Contemporary Math.*, vol. 26, pp. 189–206, Jan. 1984.

[17] A. Joly and O. Buisson, "A posteriori multi-probe locality sensitive hashing," in *Proc. 16th Int. Conf. Multimedia*, Oct. 2008, pp. 209–218.

[18] A. Joly and O. Buisson, "Random maximum margin hashing," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, Jun. 2011, pp. 873–880.

[19] B. Kulis and T. Darrell, "Learning to hash with binary reconstructive embeddings," in *Proc. Adv. Neural Inf. Process. Syst.*, Dec. 2008, pp. 1042–1050.

[20] B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 6, pp. 2130–2137, Jun. 2012.

[21] Y. Lifshits and S. Zhang, "Combinatorial algorithms for nearest neighbors, near-duplicates and small-world design," in *Proc. ACM-SIAM Symp. Discrete Algorithms*, Jan. 2009, pp. 318–326.

[22] Y. Lin, R. Jin, D. Cai, and X. He, "Random projection with filtering for nearly duplicate search," in *Proc. 26th AAAI Conf. Artif. Intell.*, Jul. 2012, pp. 641–647.

[23] Y. Lin, R. Jin, D. Cai, S. Yan, and X. Li, "Compressed hashing," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, Jun. 2013, pp. 446–451.

[24] T. Liu, A. W. Moore, and A. Gray, "New algorithms for efficient high dimensional non-parametric classification," *J. Mach. Learn. Res.*, vol. 7, pp. 1135–1158, 2006.

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

[26] W. Liu, J. Wang, S. Kumar, and S. F. Chang, "Hashing with graphs," in *Proc. 28th Int. Conf. Mach. Learn.*, Jun./Jul. 2011, pp. 1–8.

[27] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe LSH: Efficient indexing for high-dimensional similarity search," in *Proc. 33rd Int. Conf. Very Large Data Bases*, Sep. 2007, pp. 950–961.

[28] Y. Mu, J. Shen, and S. Yan, "Weakly-supervised hashing in kernel space," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, Jun. 2010, pp. 3344–3351.

[29] M. Muja and D. G. Lowe, "Fast approxiamte nearest neighbors with automatic algorithm configuration," in *Proc. Int. Conf. Vision Theory Applicat.*, Feb. 2009, pp. 331–340.

[30] M. Muja and D. Lowe, *FLANN: Fast Library for Approximate Nearest Neighbors User Manual*, [Online]. Available: http://www.cs.ubc.ca/research/flann/uploads/FLANN/flann_manual-1.8.4.pdf, 2009.

[31] M. Norouzi and D. J. Fleet, "Minimal loss hashing for compact binary codes," in *Proc. 28th Int. Conf. Mach. Learn.*, Jun./Jul. 2011, pp. 353–360.

[32] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *Int. J. Comput. Vision*, vol. 42, no. 3, pp. 145–175, 2001.

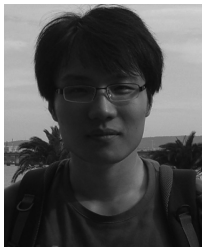
[33] R. Panigrahy, "Entropy based nearest neighbor search in high dimensions," in *Proc. 17th Annu. ACM-SIAM Symp. Discrete Algorithms*, Jan. 2006, pp. 1186–1195.

[34] R. Paredes and E. Chvez, "Using the k-nearest neighbor graph for proximity searching in metric spaces," in *Proc. 12th Int. Conf. String Process. Inform. Retrieval,* , Nov. 2005, pp. 127–138.

[35] J. P. Heo, Y. Lee, J. He, S. F. Chang, and S. E. Yoon, "Spherical hashing," in *Proc. IEEE Conf. Comput. Vision Pattern Recogn.*, Jun. 2012, pp. 2957–2964.

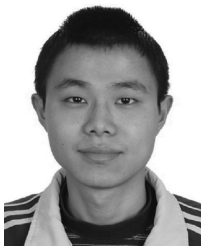
[36] M. Raginsky and S. Lazebnik, "Locality sensitive binary codes from shift-invariant kernels," in *Proc. Adv. Neural Inform. Process. Syst.*, Dec. 2009, pp. 1509–1517.

[37] S. Ray, S. Bandyopadhyay, and S. Pal, "Dynamic range-based distance measure for microarray expressions and a fast gene-ordering algorithm," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 37, no. 3, pp. 742–749, Jun. 2007.
[38] J. S. Beis and D. G. Lowe, "Shape indexing using approximate nearest-neighbour search in high-dimensional spaces," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, Jun. 1997, pp. 1000–1006.
[39] C. Silpa-Anan and R. Hartley, "Optimised KD-trees for fast image descriptor matching," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, Jun. 2008, pp. 1–8.
[40] T. Trzcinski, V. Lepetit, and P. Fua, "Thick boundaries in binary space and their influence on nearest-neighbor search," *Pattern Recognit. Lett.*, vol. 33, no. 16, pp. 2173–2180, 2012.
[41] J. Wang, S. Kumar, and S.-F. Chang, "Sequential projection learning for hashing with compact codes," in *Proc. 27th Int. Conf. Mach. Learn.*, Jun. 2010, pp. 1127–1134.
[42] J. Wang, S. Kumar, and S. F. Chang, "Semi-supervised hashing for large scale search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 12, pp. 2074–2081, Dec. 2012.
[43] J.-L. Wang and C.-Y. Chang, "Fast retrieval of electronic messages that contain mistyped words or spelling errors," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 27, no. 3, pp. 441–451, Jun. 1997.
[44] X. Wang, D. Shasha, and K. Zhang, "Metricmap: An embedding technique for processing distance-based queries in metric spaces," *IEEE Trans. Systems, Man, Cybern. B, Cybern.*, vol. 35, no. 5, pp. 973–987, Oct. 2005.
[45] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Proc. Adv. Neural Inform. Process. Syst.*, Dec. 2008, pp. 1753–1760.
[46] C. Wu, J. Zhu, D. Cai, C. Chen, and J. Bu, "Semi-supervised nonlinear hashing using bootstrap sequential projection learning," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 6, pp. 1380–1393, Jun. 2013.
[47] B. Xu, J. Bu, Y. Lin, C. Chen, X. He, and D. Cai, "Harmonious hashing," in *Proc. 23rd Int. Joint Conf. Artif. Intell.*, Aug. 2013, pp. 1820–1826.
[48] H. Xu, J. Wang, Z. Li, G. Zeng, S. Li, and N. Yu, "Complementary hashing for approximate nearest neighbor search," in *Proc. IEEE Int. Conf. Comput. Vision*, Nov. 2011, pp. 1631–1638.
[49] P. Xu, C.-H. Chang, and A. Paplinski, "Self-organizing topological tree for online vector quantization and data clustering," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 35, no. 3, pp. 515–526, Jun. 2005.
[50] M.-S. Yang and C.-H. Chen, "On the edited fuzzy k-nearest neighbor rule," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 28, no. 3, pp. 461–466, Jun. 1998.
[51] Z. Yu, D. Cai, and X. He, "Error-correcting output hashing in fast similarity search," in *Proc. 2nd Int. Conf. Internet Multimedia Comput. Service*, Dec. 2010, pp. 7–10.
[52] D. Zhang, G. Yang, Y. Hu, Z. Jin, D. Cai, and X. He, "A unified approximate nearest neighbor search scheme by combining data structure and hashing," in *Proc. 23rd Int. Joint Conf. Artif. Intell.*, Aug. 2013, pp. 681–687.
[53] S. Zhang, H.-S. Wong, Z. Yu, and H. Ip, "Hybrid associative retrieval of three-dimensional models," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 40, no. 6, pp. 1582–1595, Dec. 2010.



Zhongming Jin received the master's degree from Zhejiang University, Hangzhou, China, in 2010, where he is currently pursuing the Ph.D. degree with the State Key Laboratory of Computer-Aided Design and Computer Graphics.

His current research interests include information retrieval, computer vision, and machine learning.

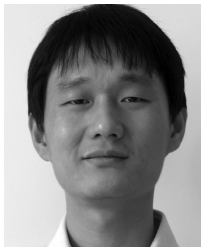


Debing Zhang received the B.S. degree in math and applied mathematics from Zhejiang University, Hangzhou, China, in 2010, where he is currently pursuing the Ph.D. degree in computer science.

His current research interests include machine learning, computer vision, and data mining.

Yao Hu received the B.S. degree in math and applied mathematics from Zhejiang University, Hangzhou, China, in 2010, where he is currently pursuing the Ph.D. degree in computer science.

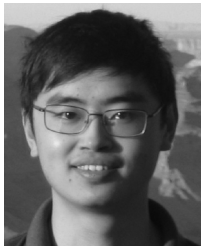
His current research interests include machine learning, computer vision, and data mining.



Shiding Lin is currently the Chief Architect of big data and infrastructure with Baidu, Inc., Beijing, China. His current research interests include distributed systems, storage systems, real-time data processing, and cloud computing.

Deng Cai (M'09) received the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign, Champaign, IL, USA, in 2009.

He is currently a Professor with the State Key Laboratory of Computer-Aided Design and Computer Graphics, College of Computer Science, Zhejiang University, Hangzhou, China. His current research interests include machine learning, data mining, and information retrieval.



Xiaofei He (M'05–SM'10) received the B.S. degree in computer science from Zhejiang University, Hangzhou, China, in 2000, and the Ph.D. degree in computer science from the University of Chicago, Chicago, IL, USA, in 2005.

He is currently a Professor with the College of Computer Science, Zhejiang University. Prior to joining Zhejiang University, he was a Research Scientist with Yahoo! Research Labs, Burbank, CA, USA.