8-2012

Boosting multi-kernel Locality-Sensitive Hashing for scalable image retrieval

Hao XIA
Nanyang Technological University

Steven C. H. HOI
Singapore Management University, chhoi@smu.edu.sg

Pengcheng WU
Nanyang Technological University

Rong JIN
Michigan State University

Citation

Boosting Multi-Kernel Locality-Sensitive Hashing for Scalable Image Retrieval

Hao Xia, Pengcheng Wu, Steven C.H. Hoi
School of Computer Engineering
Nanyang Technological University
Singapore 639798
{xiah0002,wupe0003,chhoi}@ntu.edu.sg

Rong Jin
Computer Science and Engineering Dept.
Michigan State University
East Lansing, MI, 48824
rongjin@cse.msu.edu

ABSTRACT

Similarity search is a key challenge for multimedia retrieval applications where data are usually represented in high-dimensional space. Among various algorithms proposed for similarity search in high-dimensional space, Locality-Sensitive Hashing (LSH) is the most popular one, which recently has been extended to Kernelized Locality-Sensitive Hashing (KLSH) by exploiting kernel similarity for better retrieval efficacy. Typically, KLSH works only with a single kernel, which is often limited in real-world multimedia applications, where data may originate from multiple resources or can be represented in several different forms. For example, in content-based multimedia retrieval, a variety of features can be extracted to represent contents of an image. To overcome the limitation of regular KLSH, we propose a novel Boosting Multi-Kernel Locality-Sensitive Hashing (BMKLSH) scheme that significantly boosts the retrieval performance of KLSH by making use of multiple kernels. We conduct extensive experiments for large-scale content-based image retrieval, in which encouraging results show that the proposed method outperforms the state-of-the-art techniques.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Retrieval models; H.2.8 [**Database Applications**]: Image databases

General Terms

Algorithms, Experimentation

Keywords

Image Retrieval, High-dimensional indexing, Locality-sensitive hashing, Kernel methods

1. INTRODUCTION

Similarity search, or Nearest Neighbor (NN) search, plays a critical role in Content-Based Image Retrieval (CBIR) systems [33, 24]. Typically, images in a CBIR system are represented in a high-dimensional space, and the size of an image database can easily be over millions or even billions for large-scale real-world applications. These two aspects have made CBIR an open grand challenge although it has been extensively studied for decades.

A variety of data structures have been proposed for indexing and searching data points in a low-dimensional space [31, 7, 2, 29]. These approaches work well for low dimensional data. But, when the number of dimensions grows, they often become less efficient, a phenomenon known as the *curse of dimensionality*. Specifically, the time or space requirements of these approaches often grow exponentially with the dimensionality.

Since exact NN search is hard to scale for high-dimensional data, recent studies mainly focus on *approximation* approaches [20, 14, 25, 1], which aim to remove the exponential dependence on dimensionality. Instead of finding the nearest point p to a query point q, approximate NN search allows to return any point within the distance of $(1 + \epsilon)$ times the distance from q to p. Recent studies have shown that by adopting the approximation, the complexity of NN search is reduced from exponential to polynomial in terms of its dependence on the dimensionality. Several recent studies have successfully applied the random projection idea for approximate NN search over high-dimensional data. One of the most well-known techniques in this direction is Locality-Sensitive Hashing (LSH) [20, 14, 6], which has been actively studied and successfully applied to many applications [20, 10, 1].

One limitation of regular LSH is that they require explicit vector representation of data points. Kernelized LSH (KLSH) [23] addresses this limitation by employing kernel functions to capture similarity between data points without having to know their explicit vector representation. KLSH has been shown effective empirically, but there is no formal analysis of KLSH in theory. In this paper, we first analyze the theoretical property of KLSH to better understand the behavior and capacity of KLSH in similarity search. Second, we address the limitation of KLSH. Despite the success, most existing KLSH techniques only adopt a single kernel function. This significantly limits its application to many real-world image retrieval tasks [40, 18], where images are often analyzed by a variety of feature descriptors and are measured by a wide class of diverse similarity functions.

To this end, we propose a novel Boosting Multi-Kernel Locality-Sensitive Hashing (BMKLSH) framework, which improves KLSH by learning a combination of multiple kernels. The key challenge of multi-kernel LSH is to determine an optimal combination of multiple kernels by determining appropriate bit size to each of the multiple kernels. To overcome the challenge, we propose a boosting scheme to greedily find a good solution in an efficient approach. Our extensive experiments show that BMKLSH significantly en-

hances the performance of KLSH in exploring the power of multiple kernels for CBIR.

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 gives our analysis of KLSH, and Section 4 presents the proposed BMKLSH method. Section 5 discusses our experiments. Finally Section 6 concludes this work.

2. RELATED WORK

This section reviews related work in approximate nearest neighbor (NN) search with the focus on image retrieval applications.

In literature, developing efficient techniques for indexing high dimensional images has been studied extensively in information retrieval, multimedia, and database communities [4, 5, 9]. Spatial data structure approaches (e.g., kd-tree [2, 29] or metric tree [35]) were used to handle the NN search problem; however, they scale poorly with data dimensionality. In practice, if the number of dimensions is large enough, kd-tree and other similar data structures require an expensive inspection in the data set, thereby perform no better than an exhaustive linear search that simply compares a query to every data point in the database.

Instead of solving the exact similarity search for high dimensional indexing, recent years have witnessed active studies of approximate high-dimensional indexing techniques [20, 14, 25, 3, 8, 11]. The key of most techniques is to exploit random projection to tackle the curse of dimensionality issue, such as Locality-Sensitive Hashing (LSH) [20], a very well-known and highly successful technique in this area. In general, we can group most of existing approaches into two major categories: linear projection methods and kernel-based methods. Below we briefly review the first category and then focus on discussing the second category.

For the first category, one of the most notable techniques is LSH [20], which utilizes a family of locality sensitive hashing functions which map similar items to same bucket with high probability, and dissimilar items to same bucket with low probability. With such a hash function, one can easily search within the bucket a query point belongs to in order to find nearest neighbors of the query. LSH efficiently solves the approximate similarity search problem and achieves query time in the worst case $O(dn^{1/\epsilon})$. Gionis et al. [14] improved the techniques and achieved significant query time $O(dn^{1/(1+\epsilon)})$. Recently, many studies have attempted to improve upon the regular LSH technique. For example, [25] introduced multi-probe LSH methods that reduce the space requirement of the basic LSH method. Tao et al. [34] proposed the locality-sensitive B-tree technique that mainly concerns the performance improvement of I/O disk access. Besides, there are many other existing works that accommodate LSH functions in tackling different issues, including Hamming distance [20], inner products [6], ℓ_p-norms [10], normalized partial matching [15], learned Mahalanobis metrics [27], and so on. Last but not least, there are also many recent studies of learning compact binary codes inspired by the idea of LSH [39, 16, 36, 22].

Our study is more related to the second category of kernel-based methods. In particular, kernel-based LSH (KLSH) [23] was recently proposed to overcome the limitation of the regular LSH technique that often assumes the data come from a multidimensional vector space and the underlying embedding of the data must be explicitly known and computable. KLSH provides a powerful framework to explore arbitrary kernel/similarity functions where their underlying embedding only needs to be known implicitly. In [28], the authors proposed a variant of KLSH which aims to force the expected Hamming distance between the binary codes of two vectors is related to the value of a shift-invariant kernel. In [17], the authors proposed an improved algorithm for learning binary codes

Symbol	Meaning
d	number of dimensions of input data
n	number of image examples in database
$\mathbf{X}$	$d \times n$ matrix of n image examples
$\mathbf{q}$	query image example
$\mathbf{x}_i$	i-th image example in database
K	kernel matrix
n_q	number of training query examples
m	number of different kernel functions
κ_l	l-th kernel function
b	length of a hash key
p	number of data points sampled in KLSH
t	number of indices selected in KLSH
D_t	distribution at the t-th round
T	number of boosting rounds

Table 1: Summary of notation used in this paper

with kernel to speed up the KLSH technique. Despite being studied actively, most existing kernel-based hashing methods [23, 28, 17, 26] only consider a single kernel, which cannot fully explore the potential of multiple kernel functions in a real CBIR application.

Very recently, some emerging study has attempted to apply KLSH by exploring multiple kernels. In [37], they proposed a simple solution named MKLSH by applying the existing KLSH algorithm for each specific kernel individually, and then combining the outputs of these KLSH processes. Our work however differs from their method in several aspects. First of all, their naive approach to combining multiple kernels simply treats each kernel equally, which fails to fully explore the power of combining multiple diverse kernels in KLSH. Second, their technique is essentially unsupervised, which does not fully explore the data characteristics and thus cannot achieve the optimal indexing performance. In contrast, our technique is in general supervised and is able to learn the optimal combination of multiple kernels from training data to maximize the indexing performance.

3. ANALYSIS OF KERNELIZED LSH

In this section, we first review the algorithm of Kernelized Locality-Sensitive Hashing (KLSH), and then present our analysis of KLSH.

Let $\mathbf{X} \in \mathbb{R}^{d \times n}$ be the collection of data points to be searched. Given a query $q \in \mathbb{R}^d$, to efficiently find the k nearest neighbors, LSH projects each data point into a low-dimensional binary space, referred to as the *hash* key. The hash keys are constructed by applying b binary-valued hash functions $h_1, \ldots, h_b$ to the data points in $\mathbf{X}$. KLSH generalizes LSH by introducing a kernel function $\kappa(\mathbf{x}_i, \mathbf{x}_j)$ to map a data point $\mathbf{x}_i$ to a functional space through a nonlinear feature mapping $\phi(\mathbf{x}_i)$ that satisfies the condition $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \phi^{\top}(\mathbf{x}_i)\phi(\mathbf{x}_j)$. To build the hash function, KLSH first randomly selects a subset of p data points from $\mathbf{X}$, denoted as $S = \{\mathbf{x}_1^s, \ldots, \mathbf{x}_p^s\}$, and forms a kernel matrix K over the sampled data points; it then generates b random vectors $\mathbf{e}_S^1, \ldots, \mathbf{e}_S^b$ and computes a hashing function for each random vector $\mathbf{e}_S^k$ as

$$h_k(\phi(\mathbf{x})) = \text{sign}\left(\sum_{j=1}^p w_j^k \kappa(\mathbf{x}, \mathbf{x}_j^s)\right),$$

where $\mathbf{w}^k = (w_1^k, \ldots, w_p^k)^{\top}$ is given by $\mathbf{w}^k = K^{-1/2}\mathbf{e}_S^k$. Algorithm 1 outlines the key steps of KLSH, where b is a critical parameter that determines the length of hash key to be constructed in KLSH.

Although KLSH is proved to be effective empirically, no theoretical analysis is provided for the properties of KLSH. To bound approximation error of KLSH, we introduce a few notations. De-

Algorithm 1 KLSH

INPUT:
- an image database: $\{\mathbf{x}_i | i = 1, \ldots, n\}$, $\mathbf{x}_i \in \mathbb{R}^d$
- length of the hash key: b
- a kernel function: $\kappa(\cdot, \cdot)$

1: randomly select p data points $S = \{\mathbf{x}_j^s | j = 1, \ldots, p\}$
2: $K = [\kappa(\mathbf{x}_i^s, \mathbf{x}_j^s)]_{p \times p}$
3: **for** $k = 1, \ldots, b$ **do**
4:     form $\mathbf{e}_S^k$: select t indices at random from $[1, \ldots, p]$
5:     form $\mathbf{w}^k = K^{-1/2} \mathbf{e}_S^k$
6:     construct hash function:
       $h_k(\phi(\mathbf{x})) = \text{sign}(\sum_{j=1}^p w_j^k \kappa(\mathbf{x}, \mathbf{x}_j^s))$
7: **end for**

OUTPUT: a set of b hash functions $\mathcal{H} = \{h_k | k = 1, \ldots, b\}$

---

fine $\mathbf{h}_i \in \mathbb{R}^b$ as

$$\mathbf{h}_i = \left(\sum_{j=1}^p w_j^1 \kappa(\mathbf{x}_i, \mathbf{x}_j^s), \ldots, \sum_{j=1}^p w_j^b \kappa(\mathbf{x}_i, \mathbf{x}_j^s) \right)^\top$$

and $H = (\mathbf{h}_1, \ldots, \mathbf{h}_n)$. Let $K_a = [\kappa(\mathbf{x}_i, \mathbf{x}_j)]_{n \times n}$ be the Gram matrix for all data points in $\mathbf{X}$, and $K_b = [\kappa(\mathbf{x}_i, \mathbf{x}_j^s)]_{n \times p}$ be the kernel matrix between data points in $\mathbf{X}$ and the sampled data points in S. Let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of K_a ranked in the descending order. We first bound the error caused by the random vectors $\{\mathbf{e}_S^k\}_{k=1}^b$.

THEOREM 1. *With a probability at least $1 - \delta$, we have*

$$\left| \frac{t}{p} K_b K^{-1} K_b^\top - \frac{1}{b} H^\top H \right|_F \leq \sqrt{\frac{2t \ln(2/\delta)}{b}} |K_b K^{-1} K_b^\top|_F$$

PROOF. Let us define A as $A = \frac{1}{b} \sum_{k=1}^b \mathbf{e}_S^k [\mathbf{e}_S^k]^\top$. It is easy to see that $\mathrm{E}[A] = \frac{t}{p} I$. According to the concentration inequality of linear operator [32], we have, with a probability $1 - \delta$, that

$$\left| A - \frac{t}{p} I \right|_F \leq \sqrt{\frac{2t \ln(2/\delta)}{p}}$$

We complete the proof by the using the fact

$$\left| \frac{t}{p} K_b K^{-1} K_b^\top - \frac{1}{b} H^\top H \right|_F = \left| K_b K^{-1/2} \left(\frac{t}{p} I - A \right) K^{-1/2} K_b^\top \right|_F$$

$$\leq \left| \frac{t}{p} I - A \right|_F |K_b K^{-1} K_b^\top|_F$$

$\square$

THEOREM 2. *Assume $\kappa(\mathbf{x}, \mathbf{x}) \leq 1$ for any $\mathbf{x}$. Let $k \in \mathbb{Z}$ and $\epsilon \in (0, 1)$ be any two numbers satisfying $k/\epsilon^4 \leq p/[64\eta^2]$, where $\eta = 1 + \sqrt{8 \log(1/\delta)}$. Then, with a probability at least $1 - 2\delta$, we have*

$$\frac{1}{n^2} \sum_{i,j=1}^n \left(\kappa(\mathbf{x}_i, \mathbf{x}_j) - \frac{p}{tb} \mathbf{h}_i^\top \mathbf{h}_j \right)^2 \leq \frac{3}{n} \sum_{i=1}^{n-k} \lambda_{k+i}^2 + 3\epsilon + \frac{6p^2 \ln(2/\delta)}{bt}$$

PROOF. According to Theorem [12], with a probability $1 - \delta$, for any $\epsilon \in [0, 1]$ and $k \in \mathbb{Z}$ that satisfy

$$\frac{k}{\epsilon^4} \leq \frac{p}{64\eta^2}$$

where $\eta = 1 + \sqrt{8 \log(1/\delta)}$, we have

$$|K_a - K_b K^{-1} K_b^\top|_F \leq \sqrt{\sum_{i=1}^{n-k} \lambda_{k+i}^2} + \epsilon n$$

where $| \cdot |_F$ stands for the Frobenius norm. Using the triangle inequality of Frobenius norm, we have

$$\left| \frac{t}{p} K_a - \frac{1}{b} H^\top H \right|_F$$

$$\leq \frac{t}{p} \left| K_a - K_b K^{-1} K_b^\top \right|_F + \left| \frac{t}{p} K_b K^{-1} K_b^\top - \frac{1}{b} H^\top H \right|_F$$

Combining the result from Theorem 1, we have, with a probability $1 - 2\delta$,

$$\left| \frac{t}{p} K_a - \frac{1}{b} H^\top H \right|_F$$

$$\leq \frac{t}{p} \sum_{i=1}^{n-k} \lambda_{k+i} + \frac{t}{p} \epsilon n + \sqrt{\frac{2t \ln(2/\delta)}{b}} |K_b K^{-1} K_b^\top|_F$$

We complete the proof by using the fact $|K_b K^{-1} K_b|_F \leq |K_a|_F \leq \sum_{i=1}^n \lambda_i \leq n$ and the definition of Frobenius. $\square$

As indicated by Theorem 2, on average, kernel similarity $\kappa(\mathbf{x}_i, \mathbf{x}_j)$ is well approximated by $\frac{p}{tb} \mathbf{h}_i^\top \mathbf{h}_j$, and the approximation error decreases as the number of bits b increases. Furthermore the approximation error is related to the eigenvalues of kernel matrix K_a. In particular, the skewed the eigenvalues of K_a, the smaller the approximation error.

Theorem 2 shows that different types of kernels will lead to very different approximation errors, depending on their eigenvalue distributions. Hence, some kernels will have good retrieval accuracy but rather poor approximation performance while the others may have the opposites. Thus, combining multiple kernels could potentially avoid relying too much on a single kernel that could be of neither poor approximation nor low retrieval accuracy. To make a good tradeoff between retrieval accuracy and approximation error, we propose a boosting scheme to combine multiple kernels for KLSH in order to obtain both high retrieval accuracy and low approximation error.

4. LEARNING TO COMBINE MULTIPLE KERNELS FOR KLSH

4.1 Overview

In this section, we propose a framework of learning the combination of multiple kernels for Kernelized Locality-Sensitive Hashing, which aims to boost KLSH by making use of a combination of multiple kernels. One key question is how to determine the weights for kernel combination. A straightforward approach is to assign equal weight to each kernel function, and apply KLSH with the uniformly combined kernel function. Such an approach might not fully explore the power of multiple kernels.

To address this limitation, we propose a scheme to assign each kernel a different number of bits so as to reflect the importance of the kernel. The key challenge of this scheme is thus how to optimize the bit size allocation with respect to different kernel functions. Figure 1 illustrates the proposed framework, which consists of two key steps: (i) bit allocation optimization, and (ii) the multi-kernel hashing as shown in Algorithm 2.
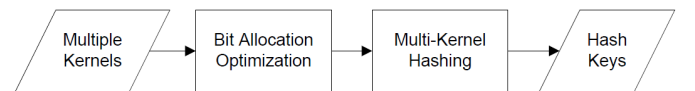

Figure 1: The proposed framework of learning to combine multiple kernels for improving kernel LSH

Algorithm 2 The Multi-Kernel Hashing scheme

INPUT:
- an image database: $\{\mathbf{x}_i | i = 1, \ldots, n\}, \mathbf{x}_i \in \mathbb{R}^d$
- a set of m different kernels: $\{\kappa_l | l = 1, \ldots, m\}$
- bits allocation vector $[b_1, \ldots, b_m]$

1: $k = 0$
2: randomly select p data points $S = \{\mathbf{x}_j^s | j = 1, \ldots, p\}$
3: **for** $l = 1, \ldots, m$ **do**
4:    $K_l = [\kappa_l(\mathbf{x}_i^s, \mathbf{x}_j^s)]_{p \times p}$
5:    **for** $r = 1, \ldots, b_l$ **do**
6:       $k = k + 1$
7:       form $\mathbf{e}_S^k$: select t indices at random from $[1, \ldots, p]$
8:       form $\mathbf{w}^k = K_l^{-1/2} \mathbf{e}_S^k$
9:       construct hash function:
         $h_k(\phi(\mathbf{x})) = \text{sign}(\sum_{j=1}^p w_j^k \kappa_l(\mathbf{x}, \mathbf{x}_j^s))$
10:    **end for**
11: **end for**
OUTPUT: a set of b hash functions $\mathcal{H} = \{h_k | k = 1, \ldots, b\}$

Unlike the regular KLSH that adopts a single kernel, BMKLSH employs a set of m kernels for the hashing scheme. Besides, a key difference between BMKLSH and some existing Multi-Kernel LSH (MKLSH) [37] is the bit allocation optimization step to find the parameter $[b_1, \ldots, b_m]$ that determines the allocations of bit sizes for a set of m kernels. Unlike the existing MKLSH approach [37] that simply assigns the same number of bits to each kernel, leading to a uniform combination of multiple kernels, we develop two supervised learning algorithms (WMKLSH and BMKLSH), described below, that effectively learn the importance of individual kernels, and consequentially determine the appropriate number of bits for each kernel.

4.2 WMKLSH by Weighted Bit Allocation

We first propose a Weighted Multi-Kernel Locality-Sensitive Hashing (WMKLSH) scheme by a supervised learning approach to determine the allocation of bit size, where a kernel is assigned a larger size of bits if it better captures the similarity between data points.

In order to learn the importance weights of different kernels for the retrieval tasks, we assume a small training data set is available for our learning task. The training set consists of a small set of queries and their relevance judgements, which usually can be easily collected in a real-life CBIR system via the relevance feedback mechanism [30, 19].

For the WMKLSH algorithm, we begin by testing the retrieval performance of KLSH with a set of m kernels on the given training set. After that, we can obtain the retrieval performance in terms of mean Average Precision (mAP), As a result, we can compute the weights based on the retrieval performance of each kernel, i.e., $\alpha_l = e^{mAP_l}, l = 1, \ldots, m$. Finally, the bit sizes of the kernels are allocated proportionally according to their importance weights.

4.3 BMKLSH for Optimizing Bit Allocation

To further improve the above learning scheme, we propose a boosting scheme, referred to as BMKLSH, to learn the optimal allocation of bit sizes, which adopts the similar idea of boosting algorithms for classification [13]. Let us denote by n_q the number of queries, and $AP_l(i), l \in [m], i \in [n_q]$, be the Average Precision performance of applying the l-th kernel $\kappa_l(\cdot, \cdot)$ to retrieve vectors for the i-th query. In order to find the optimal allocation of bit sizes, we cast it into the following optimization problem:

$$\max_{b_1, \ldots, b_m \in [b]} \sum_{i=1}^{n_q} \exp \left(\sum_{l=1}^m AP_l(i) \right) \text{subject to} \sum_l b_l = b \quad (1)$$

Since each b_l is an integer variable, the above problem is essentially an integer programming task, which is NP-hard. To find an efficient solution, we approximate the above optimization problem by introducing a set of combination weights $\alpha_l, l \in [m]$, each of which represents the importance of each kernel so as to determine each bit size b_i. In order to learn the optimal weights, we turn the above optimization into the following optimization problem

$$\max_{\alpha \in \mathbb{R}_+^m, \alpha^\top \mathbf{1} = 1} \sum_{i=1}^{n_q} \exp \left(\sum_{l=1}^m \alpha_l AP_l(i) \right) \quad (2)$$

Given the learned weights α, we assign to the l-th kernel $b\alpha_l$ bits, where b is the total number of bits.

To efficiently solve the problem in (2), we adopt a boosting based strategy. Following the similar procedure of Adaboost algorithm [13], we introduce a distribution of weights D_t to indicate the retrieval difficulty of the instances in the training data set. At each boosting round, we measure the retrieval performance (e.g. average precision) of each kernel based on the existing KLSH algorithm (shown in Algorithm 1), and select the best kernel with the largest weighted Average Precision (wAP) performance over the current distribution D_t, which is defined as:

$$wAP_l = \sum_{i=1}^{n_q} D_t(i) AP_l(i) \quad (3)$$

At the end of each boosting round, an importance weight is computed for the selected kernel based on its retrieval performance, and the weights of each poorly retrieved query example will be increased such that the next selected kernel will focus more on those hardly retrieved examples. The boosting procedure will be repeated T times. Finally, we allocate a bit size to each kernel based on its cumulative weight in all the T boosting rounds. The details of the proposed algorithm are given in Algorithm 3.

4.4 Time Complexity Analysis

First of all, assume the length of hash key b is fixed, both WMKLSH and BMKLSH algorithms have the same querying time cost as that of KLSH and MKLSH. Next we focus on discussing training and indexing time cost.

As supervised methods, WMKLSH and BMKLSH algorithms need to test the performance for all training instances with all kinds of kernels once. For BMKLSH, at each boosting round, it only updates the distribution of training instances and computes the weight for each round based on the distribution. The updating step is in general linear. Thus, the main time cost consumed mostly falls into the process of validating the performance of training instances.

5. EXPERIMENTS

We conduct extensive experiments to examine the efficacy of the proposed algorithms for scalable content-based image retrieval.

5.1 Experimental Testbed

We perform experiments on two well-known public image databases which have been widely used for benchmark image retrieval tasks. Besides, we downloaded 1,000,000 social images from Flickr to form a background class and combine it with the second database to form a large scale data set. We briefly introduce some details of the two data sets below.

The first data set is the INRIA Holidays data set[1], which has been widely used for benchmark evaluation of image retrieval performance [21]. It contains 500 image groups, each of which repre-

---
[1] http://lear.inrialpes.fr/~jegou/data.php

Algorithm 3 BMKLSH: Boosting Multi-Kernel LSH algorithm for optimizing the bit size allocation

INPUT:
- an image database: $\{\mathbf{x}_i | i = 1, \ldots, n\}, \mathbf{x}_i \in \mathbb{R}^d$
- length of the hash key: b
- a set of m different kernels, $\{\kappa_l | l = 1, \ldots, m\}$
- a set of n_q training instances $\{\mathbf{x}_i^t | i = 1, \ldots, n_q\}$ with additional feedback info (positive and negative lists)
- number of boosting rounds: T
- initial distribution: $D_1(i) = 1/n_q, i = 1, \ldots, n_q$

1: **for** $t = 1, \ldots, T$ **do**
2:     **for** $l = 1, \ldots, m$ **do**
3:         obtain $AP_l(i), l = 1, \ldots, m, i = 1, \ldots, n_q$ by testing on the training set based on Algorithm 1
4:         $wAP_l = \sum_{i=1}^{n_q} D_t(i) AP_l(i)$
5:         $\alpha_t^l = e^{wAP_l}$
6:     **end for**
7:     $\alpha_t^l \leftarrow \frac{\alpha_t^l}{\sum_{i=1}^m \alpha_t^i}, l = 1, \ldots, m$
8:     $l^* = \arg \max_{l \in \{1, 2, \ldots, m\}} \alpha_t^l$
9:     $\alpha_t \leftarrow \alpha_t^{l^*}, AP_t \leftarrow AP_{l*}, wAP_t \leftarrow wAP_{l*}$
10:    update the distribution of the training instances:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & if \; AP_t(i) \geq wAP_t \\ e^{\alpha_t} & if \; AP_t(i) < wAP_t \end{cases}$$

     Z_t is a normalization factor to make D_{t+1} a distribution
11: **end for**
12: $\alpha_l = \sum_{t=1}^T \alpha_t^l, l = 1, \ldots, m$
13: allocate bits to m kernels $[b_1, \ldots, b_m]$ based on the weights α_l
14: Multi-Kernel Hashing($[b_1, \ldots, b_m]$)

---

sents a distinct scene or object. The first image of each group is the query image and the correct retrieval results are the other images of the group. There are 1491 images in total, including 500 queries and 991 corresponding relevant images.

The second data set is a large-scale data set, which consists of both the ImageCLEF database[2] and the collection of 1,000,000 images crawled from Flickr (named as "Flickr1M"). ImageCLEF is a medical image database, which was also used in [38]. For all the categories, we randomly select 10% images as the query pool, the other images as database pool. For the Flickr photos, we treat all of them as the background noisy photos, which are mainly used to test the scalability of our algorithms. We denote this data set as the "ImageCLEF + Flickr1M" data set or "ImageCLEFFlickr" for short.

5.2 Experimental Setup

We present our experimental results by showing the percentage of database items searched with the hashing function instead of measuring the exact search time in order to avoid the unfairness from different implementations of the codes. To achieve this purpose, we have to set a parameter for the LSH search, i.e., ρ, which is used to control the fraction of nearest neighbors to be linearly scanned.

For performance metric, we evaluate the retrieval performance based on mean Average Precision (mAP) and top-n ($n = 1, 2, \ldots, 5$) retrieval accuracy. The Average Precision (AP) value is the area under precision-recall curve for a query. The mAP value is calculated based on the average AP value of all the queries. The precision value is the ratio of relevant examples over the total retrieved examples, while recall is the ratio of the relevant examples retrieved over the total relevant examples in the database.

Our objective is to achieve fast and accurate image retrieval. As LSH can only return a portion of images, only the top returned images (Hit Items) are needed to be evaluated. Based on this concern, the mAP performance reported in our experiments is based on the mAP over all the returned items (except for the experiments of parameter sensitivity evaluation, where we adopt the top-10 mAP performance to enable a fair evaluation of different parameter settings). It is worth mentioning that in some previous works they usually reported mAP values for the whole data set. As we know, the more the image examples retrieved the higher the mAP value obtained. Thus, it is not totally fair to directly compare the exact value of our results with the results of the previous works. In our experiments, we have implemented all the compared methods including our algorithms under the same evaluation criterion to enable fair comparisons. Nonetheless, from the experimental results, we found that the mAP results we achieved are higher than some of the others's reported results, although we are actually based on a criterion that generates relatively lower mAP values.

As the proposed algorithms need some training data, we split the original query set into two parts, each time we select one part for query, and the other part for training. The final result is obtained by computing the average of the results over the two splits. Finally, we conduct the evaluations of all the algorithms 10 times and average the results over these 10 runs to obtain a stable result.

5.3 Image Descriptors and Kernel Functions

5.3.1 Image Descriptors

We adopt both global and local feature descriptors for representing images in our experiments. We have some preprocessing by resizing all images to 500×500 pixels while keeping the aspect ratio unchanged. For global features, we extract five kinds of features, including (1) Color histogram and color moments, (2) Edge direction histogram, (3) Gabor wavelets transform, (4) Local Binary Pattern, and (5) GIST. For local features, we extract the bag of visual words features based on two types of descriptors: SIFT and SURF. In particular, for SIFT: we adopt the Hessian-Affine interest region detector with threshold 500 and the SIFT descriptor; for SURF: we use SURF detector with threshold 500 and SURF descriptor. For the clustering, we adopt a forest of 16 kd-trees and search 2048 neighbor to speed up the clustering task. Finally, we use TF-IDF to generate the bag of visual words to represent the local features. In total, with different vocabulary sizes, we extracted four kinds of local features, including SIFT200, SIFT1000, SURF200 and SURF1000.

5.3.2 Kernel Functions

From the above, we represent each image by 9 types of different features. We then build kernel functions for these 9 types of features. Before we compute the kernels, we normalize the feature vectors to zero mean (each dimension) and unit length (each point). We adopt the RBF kernel:

$$\kappa(\mathbf{x}, \mathbf{x}') = exp(-\gamma^{-1} d(\mathbf{x}, \mathbf{x}')) \tag{4}$$

where $d(\cdot, \cdot)$ is the distance and γ is selected as the mean of the pairwise distance where the distance is computed using L2 distance. In total, we have a set of 9 kernels for our retrieval tasks. Finally, we note that all kernel matrices are normalized to unit trace to balance different kernels.

5.4 Comparison Algorithms

To extensively examine the efficacy of the proposed algorithms, we have implemented several different solutions for adopting KLSH
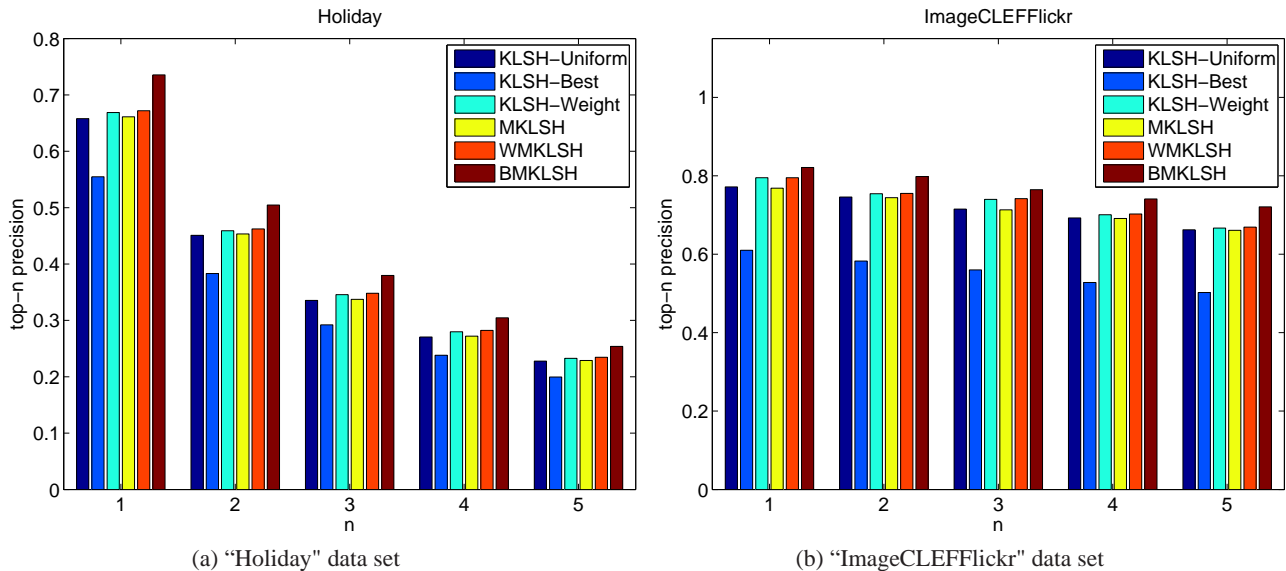
(a) "Holiday" data set   (b) "ImageCLEFFlickr" data set

Figure 2: Evaluation of average top-n precision of retrieval results by different algorithms.

with multiple kernels. In particular, we have implemented the following algorithms:

- KLSH-Uniform: a baseline method that uniformly combines the m kernels, i.e., $\kappa = \sum_{l=1}^{m} \frac{1}{m}\kappa_l$, and adopts this combined kernel for KLSH.

- KLSH-Best: We test the retrieval performance of all kernels, evaluate their mAP values on the training set, and then select the best kernel (with the highest mAP value). We adopt this best kernel for KLSH.

- KLSH-Weight: We evaluate the mAP performance of all kernels on the training set, calculate the weight of each kernel w.r.t. their mAP values: $\alpha_l = e^{mAP_l}$ (the same weight function as WMKLSH and BMKLSH), and finally normalize the weights (sum to be 1). Finally, we adopt the weighted combination of the m kernels: $\kappa = \sum_{l=1}^{m} \alpha_l \kappa_l$ for KLSH.

- MKLSH [37]: an existing KLSH approach that uses multiple kernels by a uniform bit size allocation.

- WMKLSH: the proposed algorithm by using a weighted bit size allocation for multiple kernels, as described in Section 4.2.

- BMKLSH: the proposed BMKLSH algorithm by optimizing bit size allocation via boosting as shown in Algorithm 3.

5.5 Experimental Results

We now present the performance evaluation results on the data sets. We measure the performance in terms of top-n ($n = 1, 2, \ldots, 5$) precision and the mAP value of all returned Hit items. For this experiment, we fix the parameters as follows: $\rho = 0.1$, $b = 300$, $p = 300$, $t = 30$, and $T = 20$. We will evaluate the sensitivity of these parameters in the subsequent section. We summarize the experimental results of mAP performance of the compared algorithms on the two data sets in Table 2, and illustrate the details of the top-n precision results in Figure 2. Below we discuss the empirical observations from these results.

To examine statistical significance of the comparisons, for the experimental results reported below, we highlight the best result in each group in bold font by conducting student t-tests with the significance level $\alpha = 0.05$.

Table 2: Experimental results of mAP performance.

Algorithm	Metric	Holiday	ImageCLEFFlickr
KLSH-Uniform	mean	0.58506	0.16902
	std	$\pm$ 0.00258	$\pm$ 0.00100
KLSH-Best	mean	0.50361	0.09813
	std	$\pm$ 0.00364	$\pm$ 0.00018
KLSH-Weight	mean	0.59986	0.17823
	std	$\pm$ 0.00321	$\pm$ 0.00156
MKLSH	mean	0.58994	0.16761
	std	$\pm$ 0.00110	$\pm$ 0.00086
WKLSH	mean	0.60562	0.17621
	std	$\pm$ 0.00037	$\pm$ 0.00064
BKLSH	mean	**0.66867**	**0.20460**
	std	$\pm$ **0.00337**	$\pm$ **0.00400**

5.5.1 On the "Holiday" Data Set

From the experimental result shown in Table 2 and Figure 2, we can draw several observations. First of all, by comparing the three different KLSH algorithms with different kernels, it seems a bit surprising to find that KLSH-Uniform, a simple uniform combination of all kernels, outperformed KLSH-Best, which is based on the best kernel chosen from the training set. But when thinking further, it is not difficult to explain the result as KLSH-best only explores a single kernel, while KLSH-Uniform jointly exploits multiple kernels. This result is further verified when we examine the result of KLSH-Weight, which outperform both KLSH-Best and KLSH-Uniform. These observations show that it is very important to explore the power of multiple kernels for KLSH in some real-world applications.

Second, by comparing the proposed WMKLSH and BMKLSH algorithms against the KLSH and MKLSH algorithms, we found that the proposed algorithms generally perform better than the KLSH and MKLSH algorithms, and the proposed BMKLSH algorithm attained the best result among all the compared algorithms, which was significantly better than the other algorithms. These promising results showed that the proposed BMKLSH technique is more effective to explore the power of multiple kernels for enhancing the image retrieval performance.
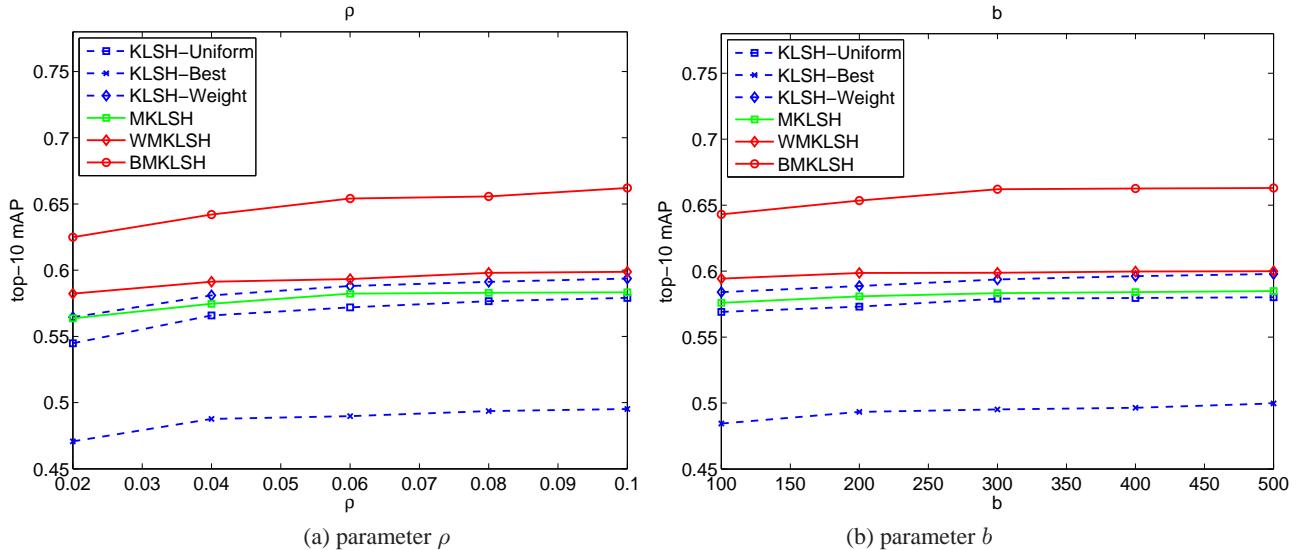
Figure 3: Evaluation of parameter ρ and b on the "Holiday" data set

5.5.2 On the "ImageCLEF+Flickr1M" Data Set

The experimental results of this data set are shown in the last column of Table 2 and the right of Figure 2. By examining the results of the three KLSH algorithms, i.e., KLSH-Uniform, KLSH-Best, KLSH-Weight, we found that the situation is the same as that on the "Holiday" data set. Further, by comparing the proposed WMKLSH and BMKLSH algorithms with the KLSH and MKLSH algorithms, we found the similar observation where the proposed BMKLSH achieved the best result among all the compared schemes. This result indicates that by the proposed boosting scheme, the BMKLSH algorithm is to effectively identify the best kernel and achieve a good tradeoff between the best kernel and the optimal combination of multiple kernels.

5.6 Parameter Sensitivity Evaluation

In this section, we aim to examine the parameter sensitivity of the proposed BMKLSH scheme for image retrieval tasks. Specifically, there are several important parameters, including (1) ρ, the parameter that controls the fraction of nearest neighbors to be linearly scanned, (2) **b**, the bit length of hash key, (3) the parameter **t** used in KLSH, which is to choose t indices for forming vector $\mathbf{e}_S$, (4) the parameter **p** used in KLSH, which is to choose a subset of p examples for computing the kernel matrix in KLSH, and (5) the number of rounds **T** used in the boosting algorithm.

For the rest of the experiments, when varying one of the parameters for evaluation, the others will remain fixed at the following default settings: $\rho = 0.1$, $b = 300$, $t = 30$, $p = 300$, and $T = 20$. We adopt the top-10 mAP performance for evaluation in this section.

5.6.1 Evaluation of ρ and b

The ρ and b are two key parameters for the LSH algorithm. Figure 3 (a) and (b) show the evaluation of two parameters ρ and b, respectively. From the experimental results, it is not difficult to see that increasing the value of ρ in general leads to increase of the top-10 mAP performance. This is not difficult to understand as the larger the ρ value, the more examples in the database will be inspected, thus more relevant image examples can be likely retrieved. Similarly, we also observe that increasing the value of hash key length b also leads to the increase of the top-10 mAP performance.

This is easy to understand as the larger the hash key length, the more information can be encoded, which thus could lead to more accurate results. Finally, similar to the previous observations, for all different values of these two parameters, BMKLSH consistently outperformed the other algorithms.

5.6.2 Evaluation of parameter p and t in KLSH

The parameter p and t are two parameters in the KLSH algorithm. Figure 4 (a) and (b) show the evaluation of two parameters p and t, respectively. From the results, we can see that increasing the p value in general leads to better performance of all the algorithms. This is reasonable as more examples are sampled we are able to obtain a more accurate estimate of the distribution for KLSH. However, when p is large enough (e.g., $p > 200$), the improvement of increasing p becomes not significant. In practice, to trade off the performance and efficacy, we can choose any value between 200 and 400 for this situation. On the other hand, for the parameter t, we found that increasing the value of t does not always lead to improvement of the performance. In some cases, a large t value could slightly degrade the performance. Nonetheless, all the algorithms are generally not very sensitive to this parameter.

5.6.3 Evaluation of the number of boosting rounds T

We now examine how the number of boosting rounds affects the performance of the proposed BMKLSH algorithm. Figure 5 shows the evaluation results. From the results, we can see that the performance of BMKLSH algorithm in general increases with respect to the increase of T. The performance of the BMKLSH algorithm becomes saturated when T is sufficiently large, e.g., $T >= 20$.

5.7 Analysis of Bit Allocation Weights

Figure 6 illustrates the bit allocation weights by three algorithms on the two data sets. The x, y, z-axis denotes the index of kernel used, the algorithm, and the weight assigned to each kernel, respectively. From this figure, we can see that MKLSH assigns equal weights to all the kernels, WMKLSH assigns different weights according to their performance and the weights are non-zero, while the weights assigned by BMKLSH are sparse, i.e., it focuses on those kernels which are more beneficial to the retrieval tasks. Specifically, for the "Holiday" dataset, the weights learned by BMKLSH
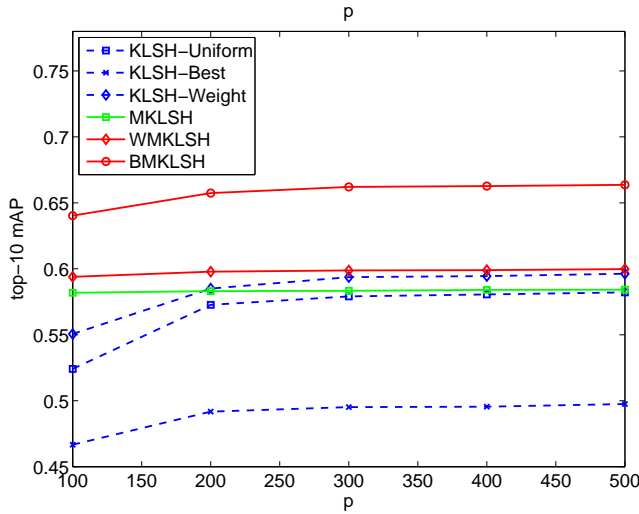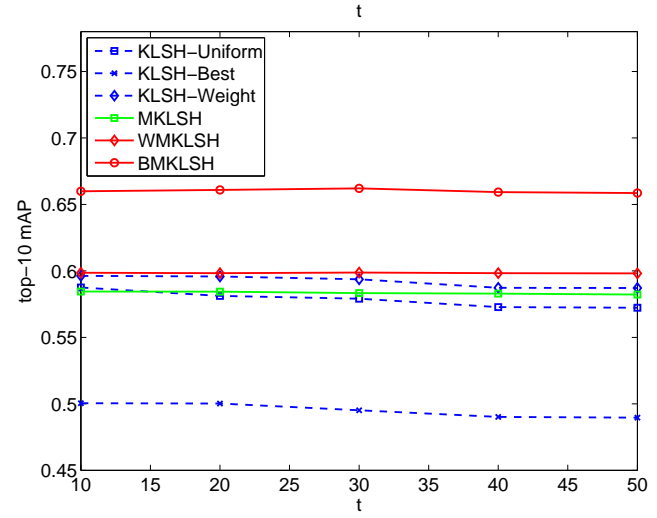
(a) parameter p (b) parameter t

Figure 4: Evaluation of two parameters p and t used in the KLSH algorithm
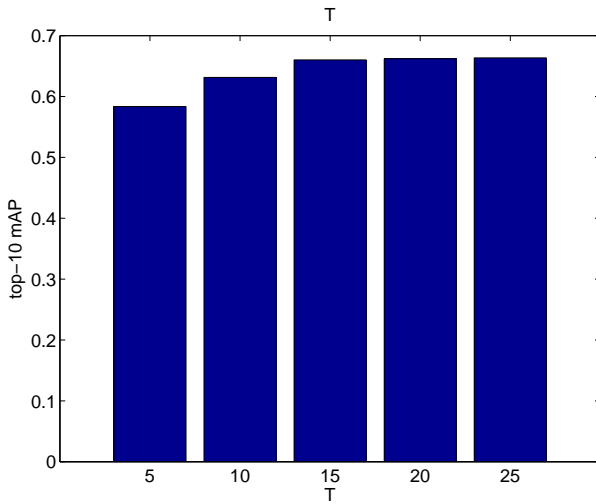
**Figure 5: Evaluation of the number of boosting rounds (T) in the proposed BMKLSH algorithm.**

are mainly assigned to Color, GIST, SIFT1000 and SURF1000; while for the "ImageCLEFFlickr" dataset, BMKLSH allocates all the weights to only GIST and SURF1000. This is reasonable as most of the medical images in the "ImageCLEF" dataset are gray-level images and contain rich texture contents, which thus favor GIST features instead of color features. Moreover, it is interesting to observe that for the "Holiday" data set, the weight of Color is less than SIFT200 and SURF200 assigned with WMKLSH, but BMKLSH filters SIFT200 and SURF200 while keeps Color. This is also quite reasonable as SIFT1000 and SURF1000 are somewhat redundant with SIFT200 and SURF200, but they are complementary to Color. These observations indicate that BMKLSH can learn an effective and sparse combination of multiple kernels.

5.8 Evaluation of Time Efficiency

Finally, we evaluate the efficiency of all the six algorithms on the "Holiday" data set. The experiments were running in Matlab on a Linux machine with 3GHz Intel CPU and 16GB RAM. As

we analyzed before, all the compared algorithms share the same querying time given a fixed bit size b. In our experiments, typically for $b = 300$, the average retrieval time per query is about 0.65ms for all the compared algorithms. In the following, we focus on the evaluation of training and indexing time efficiency.

Figure 7 shows the evaluation results of the total amount of training and indexing time cost on the holiday data set, which were averaged over 10 runs. Among all the algorithms, it is not surprising that WMKLSH and BMKLSH took more time cost for training and indexing because of the nature of their supervised learning processes. Such additional overhead is however acceptable since the training process typically is done in an offline manner. To further examine if the training and indexing process is scalable, we examine the relationship of their time cost with respect to the bit size b, and found that they generally follow a linear relationship. Besides, we also vary the number of boosting rounds T for the BMKLSH algorithm. From the results, we can see that the time cost only increases slightly w.r.t. the increase of T, which is almost neglected.
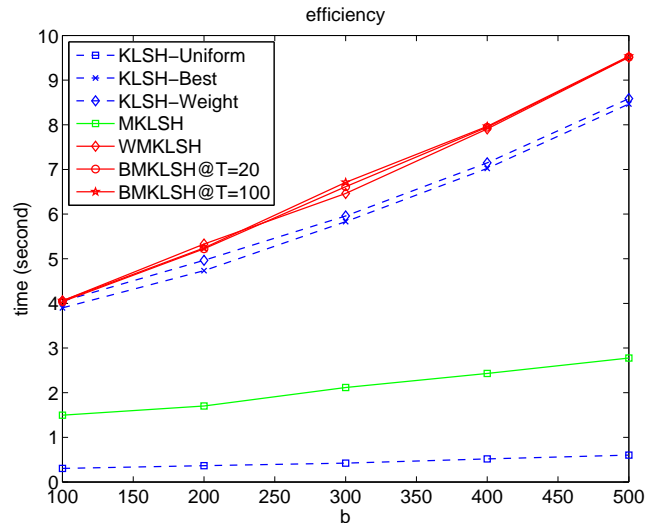
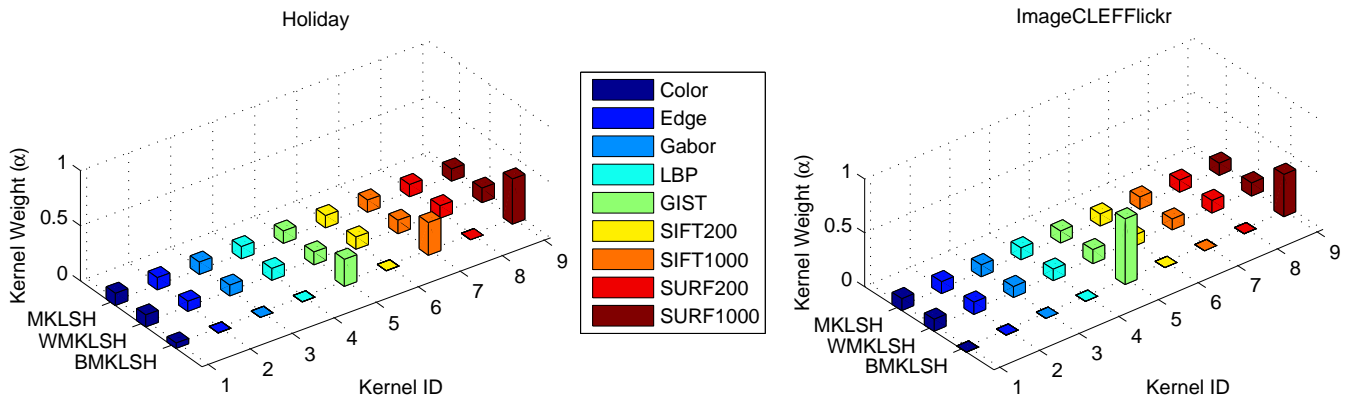**Figure 7: Evaluation of training and indexing time efficiency.**

Figure 6: Visualization of bit allocation weights obtained by three different algorithms.

5.9 Evaluation of Qualitative Performance

Finally, we illustrate the qualitative retrieval performance by randomly choosing some query images from the database. Figure 8 shows the retrieval results by different algorithms. This figure includes four retrieval cases of different queries, each of which shows the top-3 retrieved results by three representative algorithms: KLSH-Best, MKLSH, and BMKLSH. From the qualitative results, we can see that the proposed BMKLSH algorithm in general is able to return more relevant results than the other algorithms.

6. CONCLUSIONS

This paper investigated a framework of Multi-Kernel Locality-Sensitive Hashing by exploring multiple kernels for efficient and scalable content-based image retrieval. We first analyzed the theoretical property of kernel LSH (KLSH). We further emphasized that it is of crucial importance to develop a proper combination of multiple kernels for determining the bit allocation task in KLSH, although KLSH and MKLSH with naive use of multiple kernels have been proposed in literature. We thus proposed two new algorithms: (i) WMKLSH that combines multiple kernels via a simple weighted combination, and (ii) BMKLSH that employs a boosting-like scheme to optimize the bit allocation of multiple kernels for KLSH. We have conducted an extensive set of experiments to evaluate the performance of the proposed algorithms, in which the encouraging results showed that the proposed BMKLSH algorithm using the boosting approach is able to considerably surpass a number of baseline methods. Future work will apply our technique to tackle other problems, such as search-based image annotation.

Acknowledgments

7. REFERENCES

[1] S. Arya, T. Malamatos, and D. M. Mount. Space-time tradeoffs for approximate nearest neighbor searching. *J. ACM*, 57(1):1–54, 2009.

[2] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[3] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The x-tree: An index structure for high-dimensional data. In *VLDB*, pages 28–39, San Francisco, CA, USA, 1996.

[4] C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Comput. Surv.*, 33(3):322–373, 2001.

[5] G.-H. Cha, X. Zhu, P. Petkovic, and C.-W. Chung. An efficient indexing method for nearest neighbor searches in high-dirnensional image databases. *IEEE Transactions on Multimedia*, 4(1):76–87, 2002.

[6] M. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002.

[7] K. L. Clarkson. A randomized algorithm for closest-point queries. *SIAM J. Comput.*, 17(4):830–847, 1988.

[8] B. Cui, B. C. Ooi, J. Su, and K.-L. Tan. Indexing high-dimensional data for efficient in-memory similarity search. *IEEE Trans. on Knowl. and Data Eng.*, 17(3):339–353, 2005.

[9] I. Daoudi, K. Idrissi, S. E. Ouatik, A. Baskurt, and D. Aboutajdine. An efficient high-dimensional indexing method for content-based retrieval in large image databases. *Image Commun.*, 24(10):775–790, 2009.

[10] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. 20th annual symposium on Computational geometry (SCG'04)*, pages 253–262. New York, NY, 2004.

[11] W. Dong, M. Charikar, and K. Li. Asymmetric distance estimation with sketches for similarity search in high-dimensional spaces. In *SIGIR*, pages 123–130, 2008.

[12] P. Drineas and M. W. Mahoney. On the nystrom method for approximating a gram matrix for improved kernel-based learning. *Journal of Machine Learning Research*, 6:2005, 2005.

[13] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, 1997.

[14] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, 1999.

[15] K. Grauman and T. Darrell. Pyramid match hashing: Sub-linear time indexing over partial correspondences. In *CVPR*, 2007.

[16] J. He, S.-F. Chang, R. Radhakrishnan, and C. Bauer. Compact hashing with joint optimization of search accuracy and time. In *CVPR*, pages 753–760, 2011.

[17] J. He, W. Liu, and S.-F. Chang. Scalable similarity search with optimized kernel hashing. In *KDD*, pages 1129–1138, 2010.

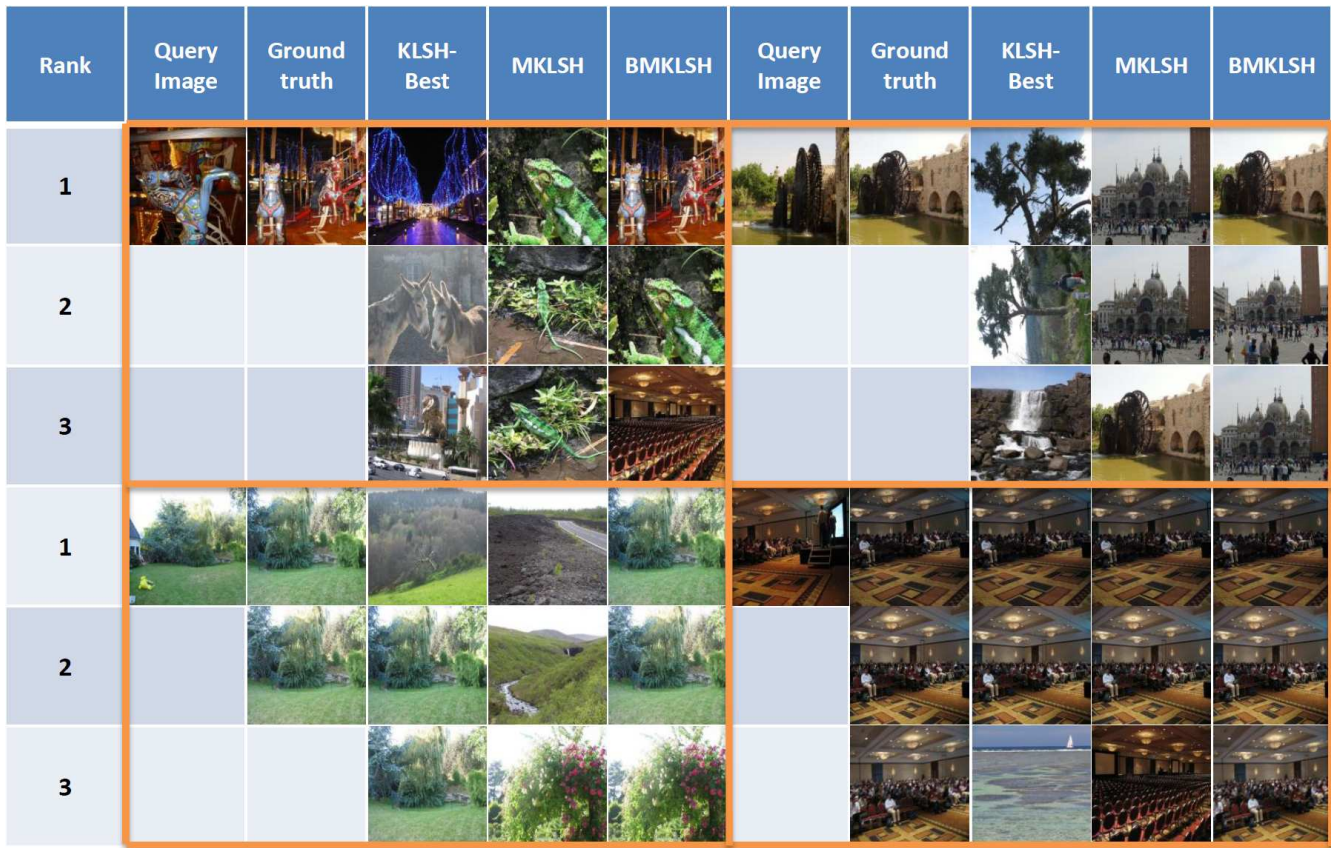[18] S. C. H. Hoi and M. R. Lyu. A multimodal and multilevel ranking

Rank	Query Image	Ground truth	KLSH-Best	MKLSH	BMKLSH	Query Image	Ground truth	KLSH-Best	MKLSH	BMKLSH
1										
2										
3										
1										
2										
3										

Figure 8: Comparison of qualitative retrieval performance on the "Holiday" dataset. This figure shows four examples. For each query, we show the top-3 retrieved images by three representative methods, i.e., KLSH-Best, MKLSH, and BMKLSH, respectively.

scheme for large-scale video retrieval. *IEEE Transactions on Multimedia*, 10(4):607–619, 2008.

[19] S. C. H. Hoi, M. R. Lyu, and R. Jin. A unified log-based relevance feedback scheme for image retrieval. *IEEE Trans. KDE*, 18(4):509–204, 2006.

[20] P. Indyk and R. Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998.

[21] H. Jegou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *ECCV (1)*, pages 304–317, 2008.

[22] A. Joly and O. Buisson. Random maximum margin hashing. In *CVPR*, pages 873–880, 2011.

[23] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*, 2009.

[24] M. S. Lew, N. Sebe, C. Djeraba, and R. Jain. Content-based multimedia information retrieval: State of the art and challenges. *ACM Trans. Multimedia Comput. Commun. Appl.*, 2(1):1–19, 2006.

[25] Q. Lv, W. Josephson, Z. Wang, M. S. Charikar, and K. Li. Multi-probe lsh: efficient indexing for high-dimensional similarity search. In *VLDB*. Vienna, Austria, 2007.

[26] Y. Mu, J. Shen, and S. Yan. Weakly-supervised hashing in kernel space. In *CVPR*, pages 3344–3351, 2010.

[27] B. K. P. Jain and K. Grauman. Fast image search for learned metrics. In *CVPR*, 2008.

[28] M. Raginsky and S. Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. In *NIPS*, pages 1509–1517, 2009.

[29] J. T. Robinson. The k-d-b-tree: A search structure for large multi-dimensional dynamic indexes. *SIGMOD*, pages 10–18, 1981.

[30] Y. Rui, T. S. Huang, M. Ortega, and S. Mehrotra. Relevance feedback: A power tool in interactive content-based image retrieval. *IEEE Trans. CSVT*, 8(5):644–655, Sept. 1998.

[31] M. Shamos and D. Hoey. Closest-point problems. In *Proc. 16th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 151–162, 1975.

[32] S. Smale and D.-X. Zhou. Geometry on probability spaces. *Constr Approx*, 30:311–323, 2009.

[33] A. W. M. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. *IEEE Trans. PAMI*, 22(12):1349–1380, 2000.

[34] Y. Tao, K. Yi, C. Sheng, and P. Kalnis. Quality and efficiency in high dimensional nearest neighbor search. In *SIGMOD Conference*, pages 563–576, 2009.

[35] J. K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40:175–179, 1991.

[36] J. Wang, O. Kumar, and S.-F. Chang. Semi-supervised hashing for scalable image retrieval. In *CVPR*, pages 3424–3431, 2010.

[37] S. Wang, S. Jiang, Q. Huang, and Q. Tian. S3mkl: scalable semi-supervised multiple kernel learning for image data mining. In *ACM Multimedia*, pages 163–172, 2010.

[38] L. Yang, R. Jin, L. B. Mummert, R. Sukthankar, A. Goode, B. Zheng, S. C. H. Hoi, and M. Satyanarayanan. A boosting framework for visuality-preserving distance metric learning and its application to medical image retrieval. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(1):30–44, 2010.

[39] D. Zhang, J. Wang, D. Cai, and J. Lu. Self-taught hashing for fast similarity search. In *SIGIR*, pages 18–25, 2010.

[40] J. Zhuang, T. Mei, S. C. H. Hoi, X.-S. Hua, and S. Li. Modeling social strength in social media community via kernel-based learning. In *ACM Multimedia*, pages 113–122, 2011.