Int J Multimed Info Retr
DOI 10.1007/s13735-014-0061-0

REGULAR PAPER

Multi-Bin search: improved large-scale content-based image retrieval

Abdelrahman Kamel · Youssef B. Mahdy · Khaled F. Hussain

Received: 16 April 2014 / Revised: 14 May 2014 / Accepted: 10 June 2014

Abstract The challenge of large-scale content-based image retrieval (CBIR) has been recently addressed by many promising approaches. In this work, a new approach that jointly optimizes the search precision and time for large-scale CBIR is presented. This is achieved using binary local image descriptors, such as BRIEF or BRISK, along with binary hashing methods, such as Locality-Sensitive Hashing and Spherical Hashing (SH). The proposed approach, named *Multi-Bin Search*, improves the retrieval precision of binary hashing methods through computing, storing and indexing the nearest neighbor bins for each bin generated from a binary hashing method. Then, the search process does not only search the targeted bin, but also it searches the nearest neighbor bins. To efficiently search inside targeted bins, a fast exhaustive-search equivalent algorithm, inspired by Norm Ordered Matching, has been used. Also, a result reranking step that increases the retrieval precision is introduced, but with a slight increase in search time. Experimental evaluations over famous benchmarking datasets (such as the University of Kentucky Benchmarking, the INRIA Holidays, and the MIRFLICKR-1M) show that the proposed approach highly improves the retrieval precision of the state-of-art binary hashing methods.

Part of this work was published in the proceedings of the 2013 IEEE 20th International Conference on Image Processing (ICIP) [17]. Source code of the proposed work can be downloaded from [11].

A. Kamel (✉) · Y. B. Mahdy · K. F. Hussain
Computer Science Department, Faculty of Computers and Information, Assiut University, Assiut 71516, Egypt
e-mail: a_kamel@fci.au.edu.eg

Y. B. Mahdy
e-mail: mahdy@aun.edu.eg

K. F. Hussain
e-mail: khussain@aun.edu.eg

Keywords Content-based image retrieval · Multi-Bin search · Binary descriptors · Binary hashing

1 Introduction

Image similarity search has been intensively addressed recently. This is due to its importance to many machine learning and computer vision applications. Such applications include object recognition [21], finding partial image duplicates on the web [28], searching individual video frames [26], image classification [30], robot localization [3], and medical imagery [13]. This work focuses on content-based image retrieval in which a large dataset of images is searched for the most similar images for a query image, it does not focus on specific object recognition or semantic queries.

Searching large-scale image datasets using brute-force matching is not a practical task. This is due to many reasons including:

- The large number of images in the dataset.
- The *high dimensionality* of image descriptors which usually consist of tens or hundreds of dimensions per descriptor.
- The method used for *matching* pairs of images.
- The complexity of the *distance measure* used to match descriptors, which is a major factor in speeding up the retrieval process and increasing the retrieval precision.

Most state-of-art large-scale image retrieval approaches use SIFT [20] feature descriptors to represent images. The SIFT descriptor has a very high discriminative power. Its drawback is the high dimensionality where a single descriptor consists of 128 dimensions. To decrease descriptors'

Published online: 03 July 2014

dimensionality, many binary descriptors have been proposed [1,4,5,18,25]. Binary descriptors are small and very fast to generate. In this work, binary descriptors are used to achieve better speedup.

To face the curse of dimensionality, many methods have been developed based on approximating the search process which is known as approximate nearest neighbor (ANN) search. Some approaches use the Bag of Visual Words or Bag of Features (BoVW or BoF) method [26] to represent images with vectors of unified dimensionality to avoid matching single descriptors inside every image. Another approach, the Vocabulary Tree [21], uses clustering algorithms, such as *k*-means, along with efficient indexing data structures to quantize descriptors and approximate the retrieval process.

Some improvements to the BoF approach are the Hamming embedding (HE) and weak geometric consistency constraints (WGC) introduced in [15,16]. HE introduces a more precise representation by providing binary signatures that refine the matching based on visual words, and WGC constraints filter matching descriptors that are not consistent in terms of angle and scale. Another major improvement to the BoF approach is the soft assignment approach [23], which explores techniques to map each visual region to a weighted set of words, allowing the inclusion of features which were lost in the quantization stage of previous systems. It generates a single descriptor for each patch as usual, but then associates that descriptor with a set of nearby clusters instead of its single nearest-neighbor cluster.

Recent approaches, such as Locality-Sensitive Hashing (LSH) [6], Spectral Hashing [27], and Spherical Hashing [10], use hashing algorithms to encode local image descriptors. These algorithms aim at encoding near neighbor image descriptors into the same binary codes. The generated binary codes, which have lower dimensionality than the original descriptors, are used to index the original descriptors into hash tables. These hash tables greatly speed up the retrieval process. In this work, some recent hashing algorithms are used to solve the problem of high dimensionality.

Quantization errors are the major problem facing quantization, clustering, and hashing methods. A query descriptor may fall on the edge of a cluster or generate a hash code that is different from its neighbors' hash codes resulting in retrieving wrong results, i.e., false positives. To solve these quantization errors, a simple and intuitive approach is proposed, in which, instead of only examining the single bin or cluster that contains the query descriptor, the nearest neighbor bins or clusters are examined to minimize quantization errors. Of course, examining more data will increase the search time, but the search accuracy would be greatly increased.

2 Related work

In this section, the work related to the proposed approach is presented starting by binary descriptors, followed by binary hashing methods, then the fast exhaustive-equivalent search algorithms.

2.1 Binary descriptors

Recent approaches to image representation favor binary descriptors such as BRISK [18], BRIEF [4], BFROST [5], ORB [25], and FREAK [1] over other high-dimensional ones such as SIFT [20] and SURF [2], this is due to their small size and fast generation. Also, the popular Hamming distance can be used directly to measure the distance between pairs of binary descriptors, this is another advantage of binary descriptors. The Hamming distance d_{H} is simply the number of different bits in a pair of binary strings, i.e., the number of ones in the binary string resulting from the bitwise XOR operation between a pair of binary strings. The number of ones in a binary string is known as the population count of the string (*popcnt*), so we may write the Hamming distance as:

$$d_{\mathrm{H}} = \mathrm{popcnt}(v_1 \oplus v_2) \tag{1}$$

In this work, the BRISK binary descriptor has been used for representing images. BRISK was proved to provide high discriminative power compared to other binary descriptors and significantly lower computational cost. The low computational cost of BRISK is due to its use of a scale-space FAST-based detector [24] in combination with a fast generation of the binary descriptor from intensity comparisons retrieved by dedicated sampling of each keypoint neighborhood.

2.2 Hashing methods

Hashing descriptors into binary codes has been proved to be efficient for solving ANN search problems. One of the most famous hashing methods is the LSH [14]. In LSH, descriptors are projected on a set of random vectors which is randomly generated from a specific distribution such as Gaussian distribution, as shown in Fig. 1a. Each random vector represents a hyperplane that works as a hash function. Each hash function generates a binary bit depending on whether a vector resides above or below the hyperplane. The total bits resulting from the hash functions constitutes the binary code. The generated binary codes may be used to build one or more hash tables or any other indexing data structure which is used in the search process.

Another efficient hashing approach is Spherical Hashing (SH) [10], in which all data vectors are projected over hypersphere-based, instead of hyperplanes, hashing func-

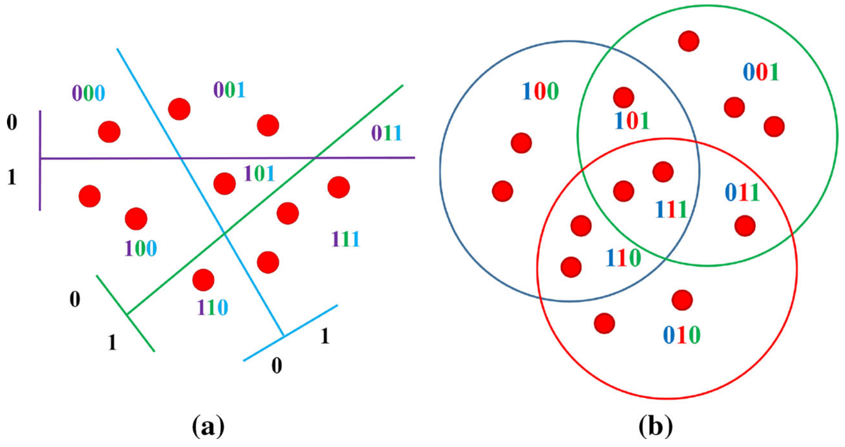

Fig. 1 Examples of **a** two-dimensional binary projections by LSH. **b** Two-dimensional spherical hashing

tions. Each hash function is represented by a hypersphere with some center vector and a radius, as shown in Fig. 1b. In Spherical Hashing, the hash functions are optimized to balance the partitioning of data and the independency between them. Unlike LSH, each spherical hash function generates a binary bit depending on whether a vector resides inside or outside a hypersphere. Spherical Hashing has been proved more efficient than most of the state-of-the-art hashing methods.

In this work, the proposed approach is applied on LSH and SH as they are considered two of the most efficient hashing methods. Evaluations show that the proposed approach highly increases the search precision of each method.

2.3 Fast exhaustive search

One step in the proposed work is to find the descriptors, inside a target bin, that are nearer to a query descriptor than others. To do this, a fast algorithm is required to examine all descriptors inside the target bin. Many exhaustive-search equivalent algorithms were introduced recently. These algorithms yield the same or too close results to exhaustive-search results with a significant speedup. Most of these algorithms depend on examining data points before matching them with a query point. Examples of these algorithms are Partial Distortion Elimination (PDE) [19], Projection Kernels (PK) [9], Low-Resolution Pruning (LRP) [8] and Norm Ordered Matching (NOM) [29].

NOM and PDE can be used with any metric distance measure, including the Hamming distance which is used in this work, where PK and LRP are designed for L_2 distance. In NOM, not all descriptors are examined to find the nearest ones, but only a group of them based on their norm values. In this work, an exhaustive-search equivalent algorithm is proposed for searching inside target bins. This algorithm is inspired by the norm matching approach of NOM.

3 Multi-Bin search

At first, it would be suitable to provide a formal definition for the problem of content-based image retrieval. Then, the approximated search process based on hashing methods is explained. Then, a Single-Bin search method, based on exhaustive-search equivalent methods is presented, that greatly increases speedup. After that, the Multi-Bin search method which can be applied on any hashing or quantization method is presented in detail. Finally, a result reranking step that increases search precision is discussed.

3.1 Formal definitions

Given a set S of images $S = \{i_1, i_2, \ldots, i_m\}$ consisting of m images and a query image q, it is required to find a set R of k images,

$$R = \{j_1, j_2, \ldots, j_k\} \quad \text{where} \quad R \subset S, \quad k \ll m \tag{2}$$

that contains the most similar k images for q in S. Similar images are those containing a visually similar scene or object with variations in illumination, viewpoint, scale, rotation, distortion, noise, etc.

To tell that two images are visually similar, firstly, a way is needed to digitally describe the visual contents of these images. This is the task of image descriptors mentioned previously, such as SIFT, SURF, BRIEF, BRISK, ORB and FREAK. Image descriptors are no more than vectors of numbers describing specific regions of an image, so an image I can be defined as a set of n vectors representing image descriptors: $I = \{v_1, v_2, \ldots, v_n\}$. Thus, replacing S by:

$$S' = \{\{v_1, \ldots, v_{n_1}\}_1, \{v_1, \ldots, v_{n_2}\}_2, \ldots, \{v_1, \ldots, v_{n_m}\}_m\} \tag{3}$$

For simplicity, S' can be replaced by:

$$S'' = \{v_{11}, \ldots, v_{1n_1}, v_{21}, \ldots, v_{2n_2}, \ldots, v_{m1}, \ldots, v_{mn_m}\} \tag{4}$$

3.2 Approximate search with binary hashing

In large-scale image retrieval, the set of images S usually consists of thousands or millions of images. Knowing that each image usually consists of a few hundreds or a few thousands of descriptors, the total number of descriptors grows to the order of billions. In this work, hashing or quantization methods are used to approximate the search process.

Any binary hashing method can be defined as follows: given a set of vectors as shown in Eq. 4, each data vector v_i is represented by a binary code $b_i = \{0, 1\}^l$, where l is the length of the binary code. To make this mapping from data vectors to binary codes, a set of l hashing functions is required

$$H(v) = (h_1(v), h_2(v), \ldots, h_l(v)) \tag{5}$$

Each hash function generates a single bit in the binary code, 0 or 1, based on specific criteria defined by the function itself. The goal of the hashing functions is to generate the same binary code for the ANN vectors.

After applying the hashing algorithm, the data vectors can be divided into a set G of *bins* or *buckets*

$$G = \{W_1, W_2, \ldots, W_{n_w}\} \tag{6}$$

each bin contains vectors with the same binary code. The maximum number of bins would be 2^l, i.e., $n_\mathrm{w} \leq 2^l$.

The approximation here is to consider all data vectors in a single bin as ANNs, i.e., matches to each other. So, the set of nearest neighbor vectors to a query vector x found inside a bin W is

$$NN(x) = \{y | y \in W_i, x \in W_i\} \tag{7}$$

Of course, the precision depends on the algorithm used for hashing and the length of the binary code. It is clear that increasing the binary code length increases the precision.

After the hashing algorithm is applied and the binary codes are generated, it is time to index all data vectors inside one or more hash tables using the binary codes as table keys.

When a query vector is submitted, its binary code is computed and used to directly access the target bin in the hash table. A step further beyond this is to search all vectors inside the target bin for the nearest neighbor vectors within a distance threshold t_v. Of course, exhaustive or brute-force search would be very time consuming. In the next subsection, a first step toward solving this problem is presented.

3.3 Single-Bin exhaustive-equivalent search (SBEES)

Exhaustively searching a target bin would be very time consuming due to the large number of vectors inside a single bin, even if the distance measure used to match vectors is very fast like the Hamming distance. So, up to now, the problem with proposed approach is the time cost of searching for near neighbor vectors inside a target bin.

To solve this problem, an exhaustive-search equivalent algorithm is proposed. Exhaustive-search equivalent algorithms yield same results as exact exhaustive-search algorithms with significant speedup. The proposed algorithm is inspired by the norm matching method introduced in NOM [29]. In the proposed algorithm, which is shown in Algorithm 1, the population count, i.e., the number of ones in the binary string, is pre-computed offline for all vectors. The algorithm computes lower and upper bounds of population count that ensure a vector may result in a distance less than the threshold. If a vector's population count is outside these bounds, it is assured that it will result in a distance greater than the threshold, so it is skipped. As a result, the distance computation are skipped for a large number of the vectors inside the bin. This algorithm is different from NOM in some steps, there is no sorting of norm values and there is no change of the upper and lower bounds. The sorting step was skipped because, in the case of binary vectors, it adds much processing overhead while skipping a small group of vectors. The upper and lower bounds are not updated because the goal of this algorithm is to find the near neighbor vectors within a constant distance threshold t_v, not the exact nearest neighbor vector as in NOM.

Computational Complexity of SBEES (Algorithm 1). Referring to Algorithm 1, let N_W be the number of descriptors in the bin W_i, and M_W be the number of descriptors in the bin W_i that achieves the condition in line 5. So, the two comparisons in line 5 are executed N_W times. Also, the distance computation step and another comparison in line 6 are executed M_W times. Assuming the descriptors consist of D 8-byte words, then the distance computation costs D XOR instructions, D POPCNT instructions, and $(D-1)$ additions. Combining all previous terms, the total cost of the algorithm is: $(2 \cdot N_W + M_W)$ comparisons, $(M_W \cdot D)$ XORs, $(M_W \cdot D)$ POPCNTs, and $(M_W \cdot (D-1))$ additions.

3.4 Multi-Bin exhaustive-equivalent search (MBEES)

Quantization errors, as discussed previously, are the major problem facing quantization, clustering, and hashing methods. This is because quantization may result in separating near neighbor descriptors into different bins or buckets, in other words, a hashing method may generate a hash code for a descriptor that is different from its nearest neighbors' hash

Algorithm 1 Searching one target bin for the nearest neighbors of a query vector using norm matching

```
Require: a query vector $x$, the target bin $W_i$ where $x \in W_i$, a set of population counts of vectors inside the bin $P_i = \{p_y | y \in W_i\}$, and a distance threshold $t_v$
Ensure: a set $NN(x)$ containing the nearest neighbor vectors of the query vector $x$ within the distance threshold $t_v$
1: $p_x \leftarrow popcnt(x)$    ▷ compute number of 1 bits in $x$
2: $lowerBound \leftarrow p_x - t_v$
3: $upperBound \leftarrow p_x + t_v$
4: for all $y | y \in W_j$ do
5:   if $lowerBound \leq p_y \leq upperBound$ then
6:     if $d_H(x, y) \leq t_v$ then
7:       Add $y$ to the set $NN(x)$
8:     end if
9:   end if
10: end for
```

codes, resulting in retrieving wrong matches, i.e., false positives, this is clarified in Fig. 2. The proposed approach aims to minimize these quantization errors; instead of only examining the single bin or bucket that contains the query descriptor, the nearest neighbor bins or clusters are also examined to minimize the quantization errors, as shown in Fig. 3. So, this approach is named *Multi-Bin Search*. Of course, there will be a trade-off between search time and accuracy as examining more data will increase both the search time and accuracy.

To apply the Multi-Bin search approach, for each bin W_i, its nearest n_{w_i} bins are *pre-computed* and stored, as shown in Algorithm 2. These nearest bins are selected based on a distance threshold t_w, which is empirically chosen, where the distance between two bins $d(W_1, W_2)$ is computed as the distance between their centers, i.e., their average vectors. This distance threshold depends, of course, on the binary code length. So, the set containing the indices of the nearest bins to some other bin w_i is

$$NN(W_i) = \{j | d(W_i, W_j) \leq t_w\} \tag{8}$$

And a bin center c_i for some bin w_i with n_i vectors is

$$c_i = \frac{\sum_1^{n_i} x_i}{n_i} \tag{9}$$

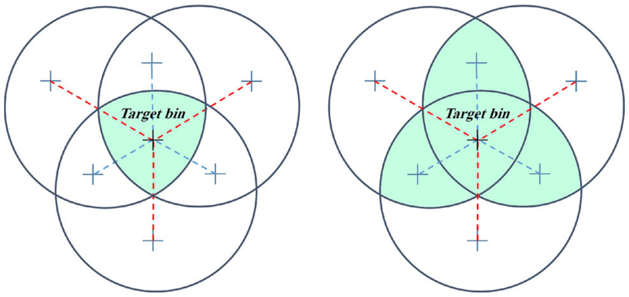

Fig. 3 *Left* In Single-Bin search, only the targeted bin is searched for ANN points (the *colored area*). *Right* In Multi-Bin search, in addition to the targeted bin, nearest neighbor bins are also searched for ANN points (the *colored area*) (color figure online)

Algorithm 2 Computing nearest neighbor bins for each bin

```
Require: bin centers $C = \{c_1, c_2, .., c_{n_w}\}$ of bins $G = \{W_1, W_2, .., W_{n_w}\}$, inter-bin distance threshold $t_w$
Ensure: a set containing the indices of the nearest neighbor bins for each bin $P = \{NN(W_1), NN(W_2), .., NN(W_{n_w})\}$ where $NN(W_i) = \{j | d(W_i, W_j) \leq t_w\}$
1: for $i = 1$ to $n_w$ do
2:   Add $i$ to the set $NN(W_i)$
3:   for $j = i + 1$ to $n_w$ do
4:     if $d(c_i, c_j) \leq t_w$ then
5:       Add $j$ to the set $NN(W_i)$
6:       Add $i$ to the set $NN(W_j)$
7:     end if
8:   end for
9: end for
```

Of course, the computation of nearest neighbor bins is done *offline* as a pre-processing step. After computing these nearest neighbor bins, they are searched, in addition to the target bin, for the nearest neighbors of the query vector x as shown in Algorithm 3. Then, only vectors within a threshold t_v, which is empirically chosen, are considered near neighbors to x.

A remaining issue with the proposed approach is how to tell that an image in the dataset is a match for a query image depending on matching individual descriptors of the query image. To resolve this issue, an algorithm is used to tell a percentage of matching between a query image and the candidate images in the dataset. For each query image descriptor, if an ANN is found, *one point* is added to the *score* of the image

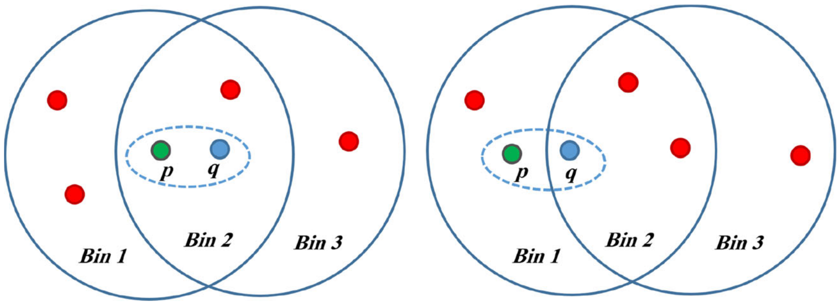

Fig. 2 *Left* A query point q and its nearest neighbor p inside the same bin or cluster. *Right* A query point q and its nearest neighbor p inside a different bin or cluster

Algorithm 3 Searching multiple nearest neighbor bins for matching vectors of a query

```
Require: query vector $x$, a set containing the indices of the nearest
    neighbor bins $NN(W_i)$ where $x \in W_i$, and a distance threshold $t_v$
Ensure: a set $NN(x)$ containing the nearest neighbor vectors of the
    query vector
1: $p_x \leftarrow popcnt(x)$
2: $lowerBound \leftarrow p_x - t_v$
3: $upperBound \leftarrow p_x + t_v$
4: for all $j | j \in NN(W_i)$ do
5:   for all $y | y \in W_j$ do
6:     if $lowerBound \le p_y \le upperBound$ then
7:       if $d_H(x, y) \le t_v$ then
8:         Add $y$ to the set $NN(x)$
9:       end if
10:    end if
11:  end for
12: end for
```

Algorithm 4 Computing image scores

```
Require: a set of images $S = \{I_1, I_2, .., I_m\}$, a set $NN(I_i) =
    \{NN(v_1), NN(v_2), .., NN(v_{n_i})$ containing the sets of the ANNs
    of each descriptor vector in the query image $I_i$, and number of top
    matched images $k$
Ensure: a set of the top $k$ matched images $R = \{I_1, I_2, .., I_k\}$
1: Let $Scores = \{s_1, s_2, .., s_m\}$
2: for $k = 1$ to $m$ do
3:   $s_k \leftarrow 0$
4: end for
5: for $i = 1$ to $n_i$ do
6:   for all $y | y \in NN(v_i)$ do
7:     $imgIndex \leftarrow j | y \in I_j$
8:     $s_{imgIndex} \leftarrow s_{imgIndex} + 1$
9:   end for
10: end for
11: Sort $Scores$ descending
12: $R \leftarrow$ first $k$ elements in $Scores$
```

to which it belongs. At the end, these scores are normalized by dividing each image's score by the sum of descriptors in the image itself and the query image. The images are sorted in descending order by their scores and the top k results are picked. This method is shown in Algorithm 4.

A major factor that significantly increases the precision of the resulting score of matching a pair of images is the actual distance between pairs of local descriptors. Of course, the shorter the distance between two descriptors is, the more similar they become. In other words, the similarity score between two descriptors is inversely proportional to the distance between them. So, it would be more precise to replace line 8 in Algorithm 4 by:

$$s_{imgIndex} \leftarrow s_{imgIndex} + \left(\frac{1}{d(v_i, y)}\right). \tag{10}$$

Computational Complexity of Algorithm 2. Referring to Algorithm 2, assuming the total number of bins is N, then the total number of iterations resulting from the two loops is $(N \cdot (N-1)/2)$. Recalling the distance computation cost from the analysis of Algorithm 1, the total cost of Algorithm 2 is: $(N \cdot (N-1)/2)$ comparisons, $(D \cdot N \cdot (N-1)/2)$ POPCNTs, $(D \cdot N \cdot (N-1)/2)$ XORs, and $((D-1)N \cdot (N-1)/2)$ additions.

Computational Complexity of MBEES (Algorithm 3*).* Referring to Algorithm 3, assume the total number of nearest neighbor bins is P, an average number of descriptors in a bin W is N_W, and the number of descriptors achieving the condition in line 6 is K. Recalling the distance computation cost from the analysis of Algorithm 1, the total cost of Algorithm 3 is: $(2P \cdot N_W + K)$ comparisons, (DK) POPCNTs, (DK) XORs, and $((D-1)K)$ additions.

Computational Complexity of Algorithm 4. Referring to Algorithm 4, assume the number of images is m, the number of descriptors in the query image is U and the average number of nearest neighbors of each descriptor is V. Then, the computational cost of Algorithm 4 is: (UV) additions and $(m \cdot log(m))$ comparisons for sorting scores.

3.5 Reranking results

A result reranking method is presented, in which the top k retrieved images are exhaustively re-matched with the query image. This method depends on approximating the exhaustive matching of a pair of images. Exhaustively matching a pair of images, say I_1 and I_2, using their individual descriptors requires finding the *exact* nearest neighbor of each descriptor in I_1 from all descriptors in I_2 and vice versa. This is a time-consuming method, so it is better to approximate it using some threshold t and searching for the *first* descriptor in I_2 which is at distance $d \le t_v$ from each descriptor in I_1. Once a match descriptor found, the rest is skipped.

This algorithm does not go backward, it only matches the descriptors from one image to the other, as shown in Algorithm 5. This algorithm also uses the norm matching method, discussed earlier in the text, for speeding the search for matched descriptors. The results will show that this approximated reranking method highly increases retrieval precision while costing constant and small amount of time per single query.

Computational Complexity of Algorithm 5. Referring to Algorithm 5, assume the number of descriptors in the first image is n, the number of descriptors in the second image is m, and the number of descriptors achieving the condition in line 6 is K. Recalling the computational cost of Algorithm 1, the computational cost of Algorithm 5 is: $(2nm + K)$ comparisons, (KD) XORs, (KD) POPCNTs, $(KD + 1)$ additions.

Algorithm 5 Approximated matching algorithm for matching a pair of images

```
Require: two sets of image descriptor vectors $I_1 = \{v_{11}, v_{12}, .., v_{1n}\}$, $I_2 = \{v_{21}, v_{22}, .., v_{2m}\}$ where $n \geq m$, their corresponding population counts $P_1 = \{p_{11}, p_{12}, .., p_{1n}\}$, $P_2 = \{p_{21}, p_{22}, .., p_{2m}\}$, and a distance threshold $t_v$
Ensure: a score $s$ representing the matching percentage between $I_1$ and $I_2$
1: $s \leftarrow 0$
2: for $i = 1$ to $n$ do
3:   $lowerBound \leftarrow p_{1i} - t_v$
4:   $upperBound \leftarrow p_{1i} + t_v$
5:   for $j = 1$ to $m$ do
6:     if $lowerBound \leq p_{2j} \leq upperBound$ then
7:       if $d_H(v_{1i}, v_{2j}) \leq t_v$ then
8:         $s \leftarrow s + 1$
9:       end if
10:    end if
11:  end for
12: end for
13: $s \leftarrow s/(n + m)$
```

4 Results and discussion

This section presents the evaluation protocol of the proposed approach and the benchmarking dataset. Then, experimental evaluation results are presented and discussed thoroughly.

4.1 Evaluation protocol

The proposed approach has been evaluated against three famous datasets:

- The University of Kentucky Benchmarking (UKB) [21]: consists of 10,200 images grouped into 2,550 categories, each category contains four images that are considered matches to each other. Given a query image, it is required to retrieve the image itself and the other three images in the same category as the top four results. So, the precision measure is a floating-point *score* in the range from 0 to 4 representing the number of retrieved correct matches in the top four results averaged over the number of run queries.
- The INRIA Holidays [15]: a collection of 1,491 holiday images, 500 of them being used as queries. The dataset includes a very large variety of scene types (natural, man-made, water and fire effects, etc.) and images are in high resolution. The accuracy is measured by the mean Average Precision (mAP) as defined in [22].
- The MIRFLICKR-1M [12]: a collection consists of one million images downloaded from the social photography site Flickr through its public API. The images from this dataset is merged as distractors with the UKB dataset to evaluate the scalability of the proposed approach. The precision measure used in this case is the same measure used with the UKB dataset.

The proposed methods have been applied on the famous binary hashing methods, LSH and SH. Another variation of the LSH algorithm is evaluated, that requires to center the data points around the origin, i.e., zero-centering the data points, this variation is denoted LSHZC. BRISK local image descriptors were generated of size 512 bits using a detection threshold of 70 for the FAST keypoint detector, these parameters showed better recognition precision as stated in [18]. The proposed methods has been compared to the BoVW approach using the implementation of [7].

All experiments were run on an Intel®Core™i7-950 processor with 8 MB cache and 3.06 GHz clock speed, and 8 GB memory. An advantage was taken of the newly introduced POPCNT instruction which was introduced with the Nehalem microarchitecture-based Core i7 processors. This instruction efficiently computes the population count of binary strings. POPCNT has been used in measuring Hamming distances between bins or descriptors and in the offline pre-computing of the population counts for all bins and descriptors.

In all experiments, unless stated otherwise, all the 10,200 images of the UKB or the 500 queries of INRIA Holidays were run as queries against their corresponding dataset, then the average score or precision was computed. Hashing methods were tested with binary code lengths ranging from 12 to 40. Binary code lengths are chosen with the constraint that the maximum number of possibly generated bins is smaller than the total number of image descriptors; because that if the number of bins approaches the total number of descriptors, the efficiency of the approximate search approaches that of the linear search, or could be worse. For Multi-Bin search, inter-bin distance thresholds were given values as a percent of the binary code lengths, a percent of 0.125 was picked, i.e., 1 bit for each 8 bits in the binary code. This threshold empirically showed the best trade-off between search time and accuracy. In the reranking step, only the top 50 results were reranked to make a trade-off between accuracy and time. For large-scale evaluation, the UKB dataset was merged with various portions of the MIRFLICKR-1M to evaluate the scalability of the proposed approach.

All evaluated methods were denoted as follows:

- BoVW: Bag of Visual Words.
- SH: Spherical Hashing.
- LSH: Locality-Sensitive Hashing.
- LSHZC: LSH with Zero-Centered vectors.
- SBEES: Applying the Single-Bin Exhaustive Equivalent Search on some hashing method.

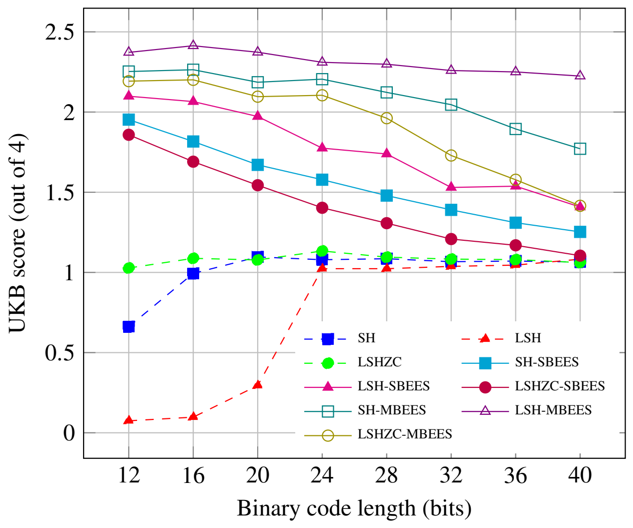

Fig. 4 Comparison of retrieval precision (UKB score) over the UKB dataset between hashing methods before and after applying SBEES and MBEES. The score is averaged over 10,200 queries. The results in this figure is slightly different from those in [17] due to some modifications in the implementation of the methods

- MBEES: Applying the Multi-Bin Exhaustive Equivalent Search on some hashing method.
- R: Applying a Reranking step.

4.2 Single-bin exhaustive-equivalent search (SBEES)

Figure 4 shows the UKB scores of the evaluated methods SH, LSH, and LSHZC, and the improvement in score resulting from applying the SBEES over the UKB dataset. These scores are averaged over 10,200 queries. It is shown that SBEES improves the retrieval precision (UKB score) of all evaluated methods by various amounts depending on binary code length. For instance, with binary code length 24, SBEES improves the retrieval precision of SH, LSH, and LSHZC by 46.24, 73.46, and 23.77 % respectively.

Table 1 shows the average query times over 10,200 queries for SH, LSH, and LSHZC before and after applying SBEES and MBEES over the UKB dataset. Of course, original hashing methods SH and LSHZC, especially with larger binary code lengths, have the shortest query times but with very low precision values as shown in Fig. 4. The LSH, LSH-SBEES, and LSH-MBEES query times are too long compared to other methods, this is due to the large sizes of bins generated by LSH. This is shown in Fig. 5a, the total number of bins generated by each hashing method for various binary code lengths. Also, Fig. 5b shows the average number of nearest neighbor bins for each bin generated by each hashing method. It is clear that LSH generates the smallest number of bins. This yields larger bins that require more time to examine. LSHZC-SBEES has the shortest times among all SBEES methods taking only 165 microseconds with binary code length of 24 bits.

4.3 Multi-bin exhaustive-equivalent search (MBEES)

Figure 4 shows the scores of the evaluated methods SH, LSH and LSHZC, and the improvement in score resulting from applying MBEES over the UKB dataset. It is shown that MBEES improves the retrieval precision (UKB score) of all evaluated methods. For instance, with binary code length 24, MBEES improves the retrieval precision of SH, LSH, and LSHZC by 104.32, 125.77, and 85.64 % respectively. This improvement in retrieval precision is gained through searching the extra neighbor bins to the target bin. The average query times over 10,200 queries for MBEES are also shown in Table 1. LSHZC-MBEES has the shortest times among

Table 1 Average query times of SBEES and MBEES compared to original hashing methods (milliseconds)

Search methods	Binary code length							
	12	16	20	24	28	32	36	40
SH	7.753	1.603	0.330	0.188	0.095	0.063	0.048	0.044
LSH	154.024	119.785	44.162	3.869	2.248	0.331	0.385	0.094
LSHZC	5.979	0.770	0.222	0.070	0.045	0.093	0.040	0.038
SH-SBEES	85.644	17.257	2.875	1.324	0.527	0.314	0.148	0.118
LSH-SBEES	3,065.790	2,125.170	586.660	35.178	20.531	2.127	2.980	0.381
LSHZC-SBEES	55.656	5.723	1.230	0.165	0.136	0.055	0.059	0.113
SH-MBEES	691.121	698.419	150.290	221.746	85.964	135.536	48.338	180.063
LSH-MBEES	14,823.000	29,605.400	13,876.500	5,668.350	3,949.980	3,035.400	3,008.120	8,978.240
LSHZC-MBEES	538.413	401.179	96.911	86.607	21.601	12.317	2.705	1.556
SH-MBEES-R	738.969	745.868	196.933	268.293	131.5405	181.067	94.9854	227.83
LSH-MBEES-R	14,873.660	29,652.835	13,924.115	5,716.859	3,999.084	3,084.144	3,294.375	9,025.181
LSHZC-MBEES-R	587.607	451.363	146.307	134.680	68.847	60.160	48.852	83.299

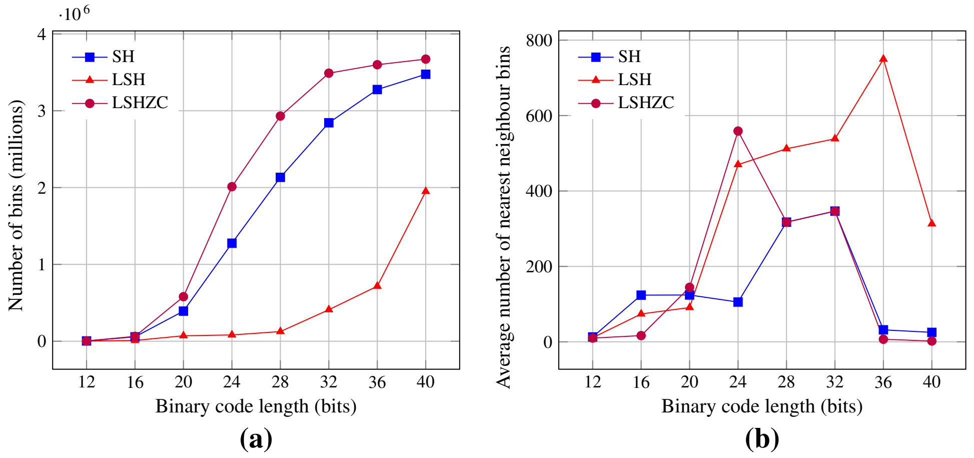

Fig. 5 **a** Total number of bins generated by each hashing method from the UKB dataset. **b** Average number of nearest neighbor bins for each bin generated by each hashing method from the UKB dataset

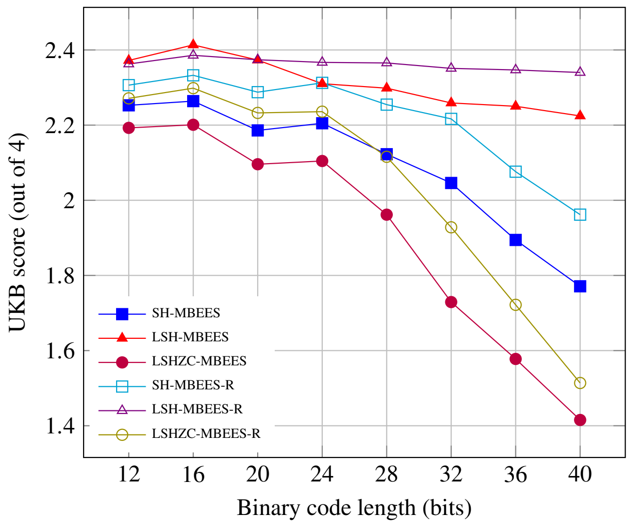

Fig. 6 Comparison of retrieval precision (average UKB score) using 10,200 queries over the UKB dataset between SH-MBEES, LSH-MBEES, and LSHZC-MBEES before and after applying the reranking step

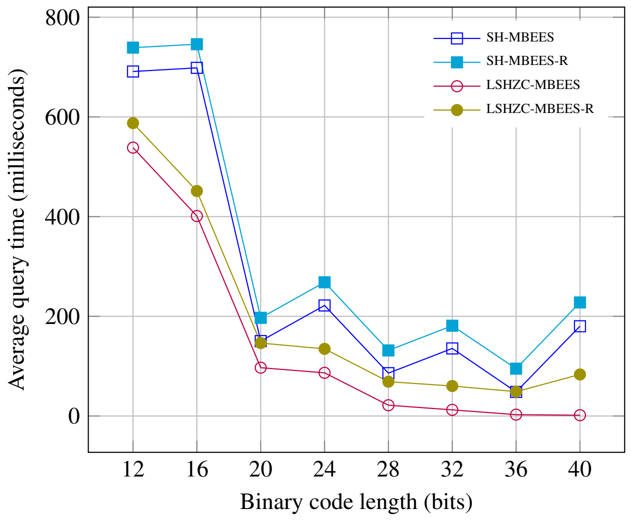

Fig. 7 Average query times using 10,200 queries over the UKB dataset before and after applying the reranking step on SH-MBEES and LSHZC-MBEES

all MBEES methods taking 86.607 milliseconds with binary code length of 24 bits.

Referring to Fig. 4, the improvement in retrieval precision resulting from MBEES decreases while increasing the binary code length. This is because increasing the binary code length leads to increasing the number of generated bins from the hashing methods, as shown in Fig. 5b. This leads to smaller bins containing smaller number of descriptors examined by the exhaustive-equivalent search.

Bringing together Fig. 4 and Table 1, it becomes clear that, for the SBEES and MBEES methods, using smaller binary code lengths in the hashing process leads to higher retrieval precisions but with longer search times, and using larger binary code lengths leads to shorter search times but with lower retrieval precisions. So, the best case for SBEES and MBEES methods is to use binary code lengths in the mid-way between the minimum and maximum allowed binary code lengths. Also, it becomes clear that SH-SBEES and SH-MBEES achieve the best trade-off between the retrieval precision and search time. Hence, SH is better suited with SBEES and MBEES than LSH and LSHZC.

4.4 Reranking results

Figure 6 shows a comparison of retrieval precision (UKB score) using 10,200 queries over the UKB dataset between MBEES before and after applying the reranking step. It shows that the reranking step increases the precision to all MBEES methods by about 10 % with increasing in query time by around 50 milliseconds. Theaverage query times of MBEES before and after applying the reranking

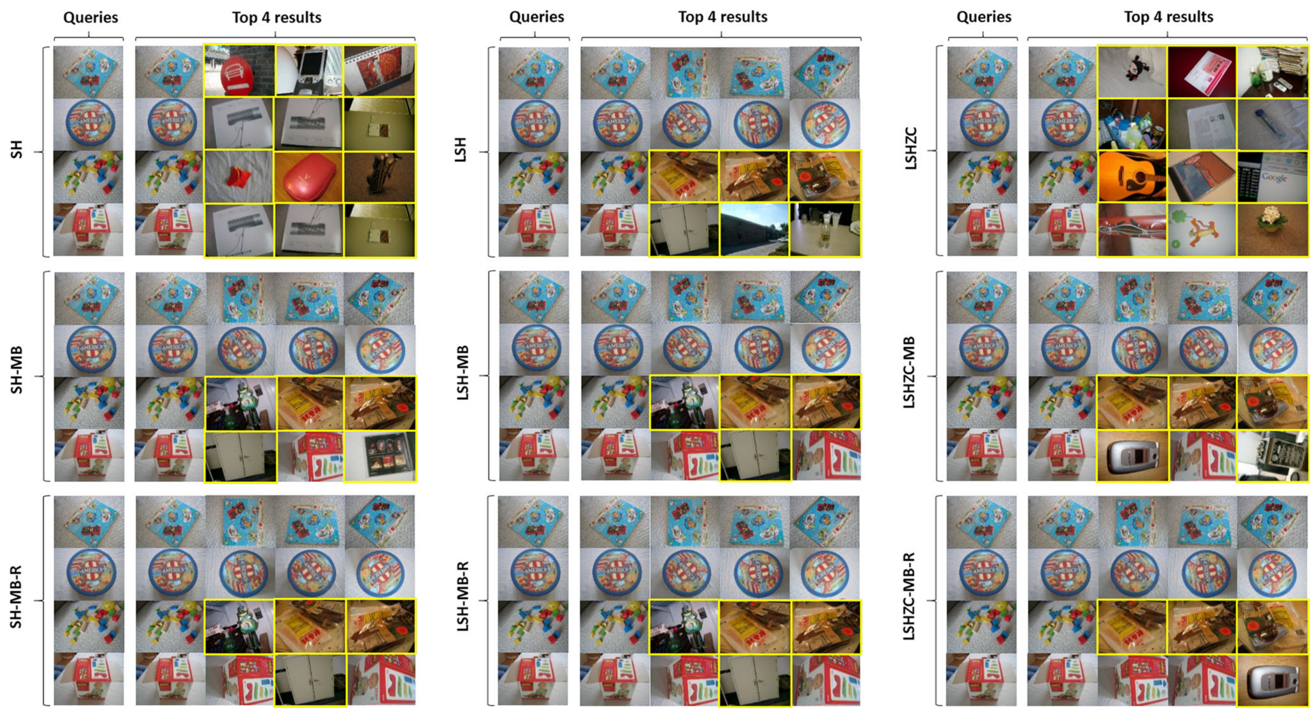

Fig. 8 Top four results of the first four distinct image queries as they appear in the UKB dataset

step is shown in Fig. 7. It is clear that SH and LSHZC gains higher increases in retrieval precision, after the reranking, than LSH while costing almost the same amounts of time.

Figure 8 shows the top four results of the first four distinct query images as found in the UKB dataset. Original LSH exceeded other original hashing methods in the first two queries due to its larger bins. MBEES improved the results of all queries except the third one, this is due to the complexity of the image's visual structure. The reranking step improved SH and LSHZC by retrieving another true positive result, i.e., another correct match.

4.5 Comparison with state-of-art methods

Table 2 shows the retrieval precision (measured in mAP) and average query times (measured in milliseconds) of the proposed methods SBEES and MBEES compared to the BoVW method over the Holidays dataset using 500 queries. It is shown that the proposed methods mostly exceeds the BoVW in precision and time, especially the SH-MBEES with binary code length 24, which achieves mAP of 42.308 at 33.793 milliseconds.

Figure 9 shows a comparison of the Recall@R (averaged over 500 queries) between SBEES, MBEES methods (using

Table 2 Retrieval precision (mAP) and average query times (milliseconds) of the proposed methods compared to the BoVW method over the Holidays dataset

Search methods	Retrieval precision (mAP)			Average query times (milliseconds)		
BoVW	($K = 6$, $L = 3$) 20.5045	($K = 10$, $L = 3$) 34.534		($K = 6$, $L = 3$) 23.9736	($K = 10$, $L = 3$) 62.345	
	(BCL = 16)	(BCL = 20)	(BCL = 24)	(BCL = 16)	(BCL = 20)	(BCL = 24)
SH-SBEES	31.840	37.095	35.515	1.766	0.484	0.201
LSH-SBEES	41.662	36.439	30.906	188.736	15.489	1.498
LSHZC-SBEES	35.364	33.422	29.233	0.906	0.167	0.077
SH-MBEES	38.209	41.973	42.308	79.178	22.177	33.793
LSH-MBEES	42.780	39.172	37.784	2,609.070	538.968	339.850
LSHZC-MBEES	40.188	41.899	39.707	55.850	9.557	12.134

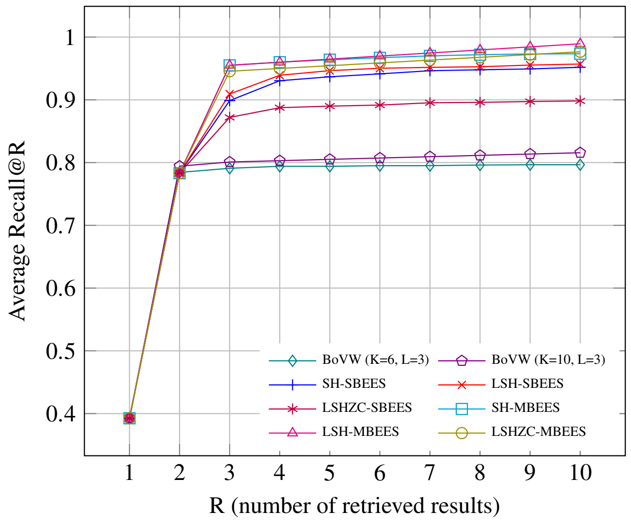

Fig. 9 Comparison of Recall@R (averaged over 500 queries) between SBEES, MBEES methods (using 24 bits) and BoVW over the Holidays dataset

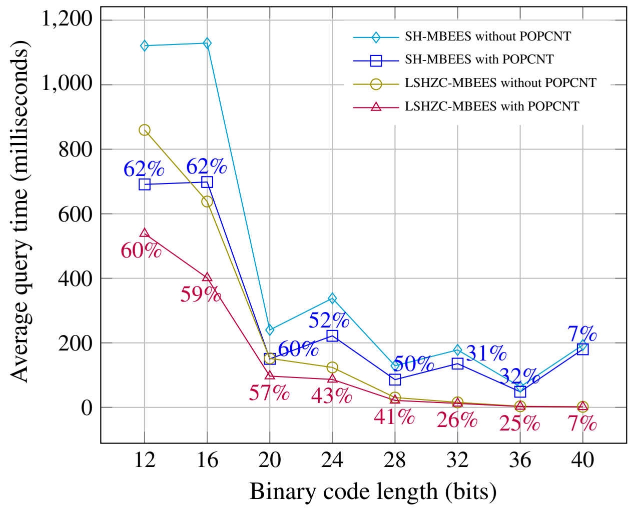

Fig. 11 Comparison between SH-MBEES and LSHZC-MBEES over the UKB dataset with and without the instruction POPCNT. The labels on the lines represents speedup

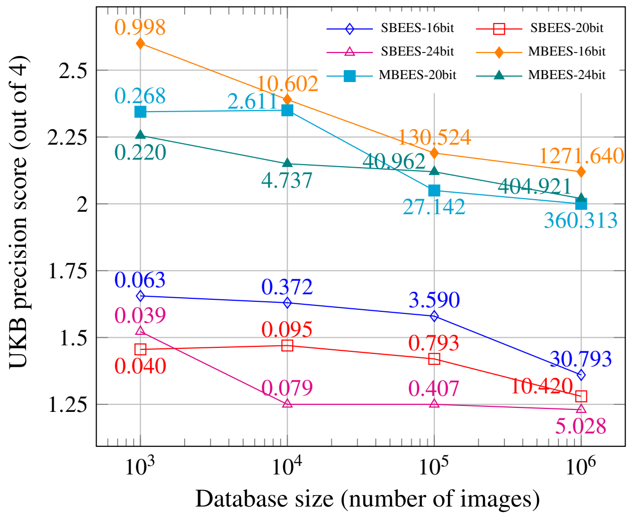

Fig. 10 UKB scores of the proposed methods applied on SH over various sizes of the MIRFLICKR-1M dataset merged with the UKB dataset. The labels on the data points represent the average query times in milliseconds

24 bits) and BoVW (with two different vocabulary sizes) over the Holidays dataset. It is shown that SBEES and MBEES achieves higher recalls, in terms of the number of retrieved items, faster than the BoVW.

4.6 Scalability evaluation

Figure 10 reveals the scalability of the proposed methods applied on SH by showing the UKB scores of the proposed methods over various sizes of the MIRFLICKR-1M dataset merged with the UKB dataset. The labels on the data points represent the average query times in milliseconds. In this experiment, only 500 queries from the UKB dataset were run as queries. Also, the number of descriptors per image was restricted to 50 descriptors per image, the most responsive descriptors were picked. It is shown that the evaluated methods are highly scalable in terms of retrieval precision and query time, especially with the 24-bit version of SH. It is clear that using smaller binary code length (16 bits) results in higher retrieval precisions but longer search times, on the other hand, using larger binary code lengths (24 bits) results in shorter search times but lower retrieval precisions. So, the best way is to use medium binary code lengths (20 bits).

4.7 Using the POPCNT instruction

Figure 11 shows the speedup improvement introduced by the POPCNT instruction, for the MBEES with SH, LSH, and LSHZC. Speedup percentages are written over each binary code length. Of course, this is not exactly the same speedup introduced only by POPCNT as there are many other computations that are not dependable on POPCNT. POPCNT is used in the query process to compute the population counts of the descriptors of the query image and in computing the Hamming distance between bins and individual image descriptors.

5 Conclusion

In this work, a large-scale image retrieval method has been proposed which is based on binary hashing methods and binary local image descriptors. In this method, not only a target bin or bucket is searched, but also the nearest neighbor bins to a target bin are searched. The nearest neighbor bins are pre-computed, then an exhaustive-search equivalent algorithm is used to examine those bins. The proposed approach can be applied to any hashing or clustering method to increase precision. The proposed approach has been applied on some hashing methods such as Spherical Hashing and LSH, the evaluations showed significant improvement in retrieval precision over the original hashing methods and in comparison with state-of-art approaches.

The work presented in this paper is a continuation to the work in [17]. The approach has been better structured and more evaluations on real-world and bigger datasets have been carried out. Also, comparison with other state-of-art methods has been presented.

Acknowledgments Thanks to the Multimedia Lab at the Multimedia Department, Faculty of Computers and Information, Assiut University, for using the computers of the lab to carry out the experiments presented in this work.

References

1. Alahi A, Ortiz R, Vandergheynst P (2012) Freak: Fast retina keypoint. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp 510–517
2. Bay H, Ess A, Tuytelaars T, Van Gool L (2008) Speeded-up robust features. Comput Vis Image Underst 110(3):346–359
3. Botterill T, Mills S, Green R (2008) Speeded-up bag-of-words algorithm for robot localisation through scene recognition. In: 23rd International Conference on Image and Vision Computing New Zealand, IVCNZ, IEEE, pp 1–6
4. Calonder M, Lepetit V, Strecha C, Fua P (2010) Brief: Binary robust independent elementary features. In: European Conference on Computer Vision (ECCV). Springer, Berlin, pp 778–792
5. Cronje J (2011) BFROST: binary features from robust orientation segment tests accelerated on the GPU. In: Proceedings of the 22nd Annual Symposium of the Pattern Recognition Society of South Africa
6. Datar M, Immorlica N, Indyk P, Mirrokni V (2004) Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of the 20th Annual Symposium on Computational Geometry. ACM, pp 253–262
7. Galvez-Lopez D, Tardos JD (2012) Bags of binary words for fast place recognition in image sequences. IEEE Trans Robot 28(5):1188–1197. doi:10.1109/TRO.2012.2197158
8. Gharavi-Alkhansari M (2001) A fast globally optimal algorithm for template matching using low-resolution pruning. IEEE Trans Image Process 10(4):526–533
9. Hel-Or Y, Hel-Or H (2005) Real-time pattern matching using projection kernels. IEEE Trans Pattern Anal Mach Intell 27(9):1430–1445
10. Heo JP, Lee Y, He J, Chang SF, Yoon SE (2012) Spherical hashing. In: International Conference on Computer Vision and Pattern Recognition (CVPR), IEEE
11. https://sites.google.com/site/multibinsearch/:. Last accessed April 2014
12. Huiskes MJ, Thomee B, Lew MS (2010) New trends and ideas in visual concept detection: the mir Flickr retrieval evaluation initiative. In: Proceedings of the International Conference on Multimedia information retrieval. ACM, pp 527–536
13. Hwang KH, Lee H, Choi D (2012) Medical image retrieval: past and present. Healthc Inf Res 18(1):3–9
14. Indyk P, Motwani R (1998) Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the 30th Annual ACM Symposium on Theory of Computing. ACM, pp 604–613
15. Jegou H, Douze M, Schmid C (2008) Hamming embedding and weak geometric consistency for large scale image search. In: European Conference on Computer Vision (ECCV). Springer, Berlin, pp 304–317
16. Jégou H, Douze M, Schmid C (2010) Improving bag-of-features for large scale image search. Int J Comput Vis 87(3):316–336
17. Kamel A, Mahdi YB, Hussain KF (2013) Multi-bin search: improved large-scale content-based image retrieval. In: 20th International Conference on Image Processing (ICIP), IEEE, pp 2597–2601. doi:10.1109/ICIP.2013.6738535
18. Leutenegger S, Chli M, Siegwart R (2011) Brisk: binary robust invariant scalable keypoints. In: International Conference on Computer Vision (ICCV), IEEE, pp. 2548–2555
19. Liu L, Shen X, Zou X (2004) An improved fast encoding algorithm for vector quantization. J Am Soc Inf Sci Technol 55(1):81–87
20. Lowe DG (2004) Distinctive image features from scale-invariant keypoints. Int J Comput Vis 60(2):91–110
21. Nister D, Stewenius H (2006) Scalable recognition with a vocabulary tree. Comput Soc Conf Comput Vis Pattern Recognit 2:2161–2168
22. Philbin J, Chum O, Isard M, Sivic J, Zisserman A (2007) Object retrieval with large vocabularies and fast spatial matching. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, pp 1–8
23. Philbin J, Chum O, Isard M, Sivic J, Zisserman A (2008) Lost in quantization: improving particular object retrieval in large scale image databases. In: International Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, pp 1–8
24. Rosten E, Drummond T (2006) Machine learning for high-speed corner detection. In: European Conference on Computer Vision (ECCV). Springer, Berlin, pp 430–443
25. Rublee E, Rabaud V, Konolige K, Bradski G (2011) Orb: an efficient alternative to sift or surf. In: International Conference on Computer Vision (ICCV), IEEE, pp 2564–2571
26. Sivic J, Zisserman A (2003) Video Google: a text retrieval approach to object matching in videos. In: International Conference on Computer Vision (ICCV), IEEE, pp 1470–1477
27. Weiss Y, Torralba A, Fergus R (2009) Spectral hashing. In: Koller D, Schuurmans D, Bengio Y, Bottou L (eds) Advances in neural information processing systems. Curran Associates, Inc., pp 1753–1760
28. Wu Z, Ke Q, Isard M, Sun J (2009) Bundling features for large scale partial-duplicate web image search. In: International Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, pp 25–32
29. Yousef M, Hussain KF (2013) Fast exhaustive-search equivalent pattern matching through norm ordering. J Vis Commun Image Represent 24(5):592–601
30. Yu L, Liu J, Xu C (2011) Descriptive local feature groups for image classification. In: 2011 18th IEEE International Conference on Image Processing (ICIP), IEEE, pp 2501–2504