Signal Processing 93 (2013) 2244–2255

Contents lists available at SciVerse ScienceDirect

Signal Processing

journal homepage: www.elsevier.com/locate/sigpro

An improved method of locality sensitive hashing for indexing large-scale and high-dimensional features

Xiaoguang Gu [a,b,c], Yongdong Zhang [a,c,*], Lei Zhang [a,b,c], Dongming Zhang [a,c], Jintao Li [a,c]

[a] Institute of Computing Technology, Chinese Academy of Sciences, No. 6 Kexueyuan South Road, Haidian District, 100190 Beijing, China
[b] Graduate University of Chinese Academy of Sciences, 19A Yuquanlu, 100049 Beijing, China
[c] Beijing Key Laboratory of Mobile Computing and Pervasive Device (Institute of Computing Technology, Chinese Academy of Sciences), No. 6 Kexueyuan South Road, Haidian District, 100190 Beijing, China

ARTICLE INFO

Article history:
Received 1 November 2011
Received in revised form
14 July 2012
Accepted 16 July 2012
Available online 25 July 2012

Keywords:
Locality sensitive hashing
Large-scale
High-dimensional
"Non-Uniform" problem
Pivot-based filtering algorithm

ABSTRACT

In recent years, Locality sensitive hashing (LSH) has been popularly used as an effective and efficient index structure of multimedia signals. LSH is originally proposed for resolving the high-dimensional approximate similarity search problem. Until now, many kinds of variations of LSH have been proposed for large-scale indexing. Much of the interest is focused on improving the query accuracy for skewed data distribution and reducing the storage space. However, when using LSH, a final filtering process based on exact similarity measure is needed. When the dataset is large-scale, the number of points to be filtered becomes large. As a result, the filtering speed becomes the bottleneck of improving the query speed when the scale of data becomes larger and larger. Furthermore, we observe a "Non-Uniform" phenomenon in the most popular Euclidean LSH which can degrade the filtering speed dramatically. In this paper, a pivot-based algorithm is proposed to improve the filtering speed by using triangle inequality to prune the search process. Furthermore, a novel method to select an optimal pivot for even larger improvement is provided. The experimental results on two open large-scale datasets show that our method can significantly improve the query speed of Euclidean LSH.

1. Introduction

Nearest Neighbors (NNs) search takes an important role in computer vision, machine learning, data mining, and information retrieval. However, for high-dimensional data, all known techniques to solve the similarity search problem will fall prey to the curse of dimensionality [1]. In some cases, such as high-dimensional learning for video annotation, the curse of dimensionality can be solved by multimodality learning [2]. In most cases, Approximate Nearest Neighbour (ANN) algorithms have been shown to be effective approaches to drastically improve the search speed while maintaining good precision. Locality sensitive hashing [3–6] is one of the most popular ANN algorithms. Until now, LSH has been successfully used for image retrieval [7,8] and 3D object indexing [9,10]. Euclidean LSH [4] is the most successful variation of basic LSH because it uses popular Euclidean distance as similarity metric. However, some distance metrics beyond Euclidean distance metric must be used in practical applications [11–14]. Therefore, some other variations are proposed for different distance metrics. Indyk and Thaper [15] have proposed the method of embedding EMD metric into the L_2 norm, then using the original LSH scheme to find the nearest neighbor in the Euclidean space. Gorisse et al. [1] present a new LSH scheme adapted to χ^2 distance for approximate nearest

* Corresponding author at: Institute of Computing Technology, Chinese Academy of Sciences, No. 6 Kexueyuan South Road, Haidian District, 100190 Beijing, China. Tel.: +86 10 6260 0666.
E-mail address: zhyd@ict.ac.cn (Y. Zhang).

0165-1684/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.sigpro.2012.07.014

neighbors search in high-dimensional spaces. Although LSH can get perfect results in theory, there are some drawbacks when using LSH in practical.

The main limitation of LSH is that its memory consumption is too large. The reason is that many hash tables are needed to keep both high recall and high precision. In [3,6], a large number of hash tables are used. When the scale of dataset is large, if the number of hash tables is also large, the index structure cannot be loaded into main memory to get the best performance. To reduce the storage space of LSH, some variations [16–18] based on multi-probe strategy have been brought forward. The first multi-probe strategy is proposed by entropy based LSH [16]. Through randomly generating neighbor points near the query point, additional hash buckets will be probed and all probe results are merged. As a result, more points will be returned and the recall will be improved. By using this method, less hash tables are needed. Multi-probe LSH [17] is inspired by and improves upon entropy based LSH and query adaptive method [19]. It proposes a more efficient algorithm to generate an optimal probe sequence of hash buckets that are likely to contain similar points to the query. Unlike multi-probe LSH which is based on likelihood criteria, posteriori multi-probe LSH [18] puts forward a more reliable posteriori model by taking account some prior knowledge about the query and the searched points. This prior knowledge helps to do a better quality control and accurately select the hash buckets to be probed. The multi-probe algorithms proposed in [16–18] can save much storage space for LSH while keeping the comparable query precision and recall. In recent years, some new hashing-based methods which convert each database item into a compact binary code are proposed to get faster query time with much less storage. Spectral hashing [20] learns data-dependent directions via principal component analysis to generate short binary codes. Wang et al. [21] propose a data-dependent projection learning method such that each hash function is designed to correct the errors made by the previous one sequentially. Motivated by Weiss et al. [20], Liu et al. [22] propose a graph-based hashing method which automatically discovers the neighborhood structure inherent in the data to learn appropriate compact codes in an unsupervised manner. Semi-Supervised Hashing (SSH) [23] is proposed to learn efficient hash codes which can handlesemantic similarity/dissimilarity among the data points. It is also much faster than existing supervised hashing methods and can be easily scaled to large datasets.

Another limitation of LSH is that the query accuracy strongly depends on the selection of parameters. LSH forest [24] is proposed to eliminate the different data-dependent parameters for which LSH must be constantly hand-tuned. LSH forest can guarantee LSH's performance for skewed data distributions while retaining the same storage and query overhead. This characteristic makes LSH forest suitable for large-scale dataset. Specially, LSH forest can be constructed in main memory, on disk, in parallel system, and in peer-to-peer systems.

LSH is efficient to index high-dimensional data. Its variations discussed above can make it index large-scale dataset. As a result, LSH has been a popular index structure for large-scale and high-dimensional dataset. However, whatever method is used, a final filtering process based on exact similarity measure is inevitable. When the scale of dataset becomes very large, the number of points needed to be filtered becomes large too. In this case, the cost of filtering is the main factor that influences the query speed. Specially, the most popular Euclidean LSH [4] uses the quantized projection of a data point on a randomly selected direction as the hash value, which makes the number of points in some buckets as significantly larger than others. At the same time, a query will also be mapped to these buckets with a high probability. This kind of phenomenon, which we call "Non-Uniform", makes the cost of filtering process significantly higher. When the data scale becomes very large, the problem will be even worse. In this work, we propose a pivot-based algorithm which uses triangle inequality to accelerate the filtering process of Euclidean LSH. The selection of pivot point can significantly influence the acceleration effectiveness. Some index structures [25–28] use some base points, which are similar to our pivots, to accelerate the search process. These base points are all selected randomly. However, random selection cannot guarantee to get the optimal result. In fact, there is no explicit criterion put forward for base point selection until now.

This paper extends our previous work [29] and has the following contributions: (1) provide a formal analysis of "Non-Uniform" problem of Euclidean LSH which can significantly decrease filtering efficiency; (2) propose a pivot-based algorithm using triangle inequality to accelerate the filtering process of Euclidean LSH; (3) propose an effective method to get an optimal pivot point which is superior to other methods. The difference between this paper and our previous work [29] lies in the following: (1) more complete analysis of related works; (2) more rigorous and aborative theoretical deduction; (3) analyse the factors that affect the performance of the proposed algorithm; (4) do more experiments on new dataset to validate the efficiency of the proposed method; (5) verify the feasibility of accelerating the proposed algorithm through sampling.

The rest of this paper is organised as follows. Section 2 introduces the background of our research. In Section 3, we propose our pivot-based filtering algorithm and the method to get an optimal pivot. Section 4 describes our experiments and Section 5 concludes this paper.

2. Background

2.1. "Non-Uniform" phenomenon of Euclidean LSH

The basic idea of LSH is to hash similar points to same bucket with higher probability than dissimilar points. Let $\mathbb{S}$ be the domain of data points and $\mathcal{D}$ be the distance measure. A function family $H = \{\mathbb{S} \to \mathbb{U}\}$ is called (r_1, r_2, p_1, p_2)-sensitive for $\mathcal{D}$ if for any $p, q \in \mathbb{S}$:

$$\text{if } \mathcal{D}(p,q) \le r_1 \quad \text{then } Pr(h(p) = h(q)) \ge p_1$$
$$\text{if } \mathcal{D}(p,q) \ge r_2 \quad \text{then } Pr(h(p) = h(q)) \le p_2$$

where $p_1 > p_2$ and $r_1 < r_2$ to ensure the function family H is useful. Hash function used in LSH is defined as:

$G = \{g : \mathbb{S} \rightarrow \mathbb{U}^k\}$

where $g(p) = (h_1(p), h_2(p), \ldots, h_k(p))$ and $h_i \in H$. The hash value of each data point is a k-dimensional integer vector and used to construct hash tables. LSH uses many hash tables to guarantee query accuracy. Different LSH schemes use different hash functions. The hash function used in the most popular Euclidean LSH is

$$h(x) = \left\lfloor \frac{\boldsymbol{a}\boldsymbol{x}+b}{W} \right\rfloor \qquad (1)$$

where W is a positive real number and b is chosen from uniform distribution $U[0,W]$; $\boldsymbol{a}$ is a vector with the same dimension number as $\boldsymbol{x}$ and each component is chosen independently from standard Gaussian distribution. Since Gaussian distribution is a 2-stable distribution [30], the distribution of $\boldsymbol{ax}/W$ is $\mathcal{N}(0, \|\boldsymbol{x}/W\|_2^2)$. Thus the probability of $\boldsymbol{ax}/W \in (-2\|\boldsymbol{x}/W\|_2, 2\|\boldsymbol{x}/W\|_2)$ is almost 95.5%. Moreover, in order to map similar points to same bucket with high probability, W is always comparable to $\|\boldsymbol{x}\|_2$, which means $\|\boldsymbol{x}/W\|_2$ is not very large. Therefore, a majority of $\boldsymbol{ax}/W$, even plus b/W (actually, the effect of plus b/W is merely translation), is distributed in a small interval (Fig. 1, left), thus after rounded to get hash value, the hash value is also distributed in a small interval. When using multi $h(x)$ to construct hash table, the number of points contained in each hash bucket will be very "Non-Uniform". Fig. 2 is the histogram of the number of points contained in each bucket of a LSH index built on ANN_SIFT1M dataset [31]. As shown in Fig. 2, a little fraction of buckets contain a large part of data points, while most buckets contain a few data points.

This phenomenon also occurs when doing a query: the hash value of query point is also distributed in a small interval (Fig. 1, right) because the hash functions are same for both the query points and the reference points. As a result, a query will also be hashed to those buckets already containing too many points with a high probability. This means that a large proportion of query points will search their neighbor points in a little fraction of buckets which contain a large number of data points. When the scale of dataset is more and more large, there

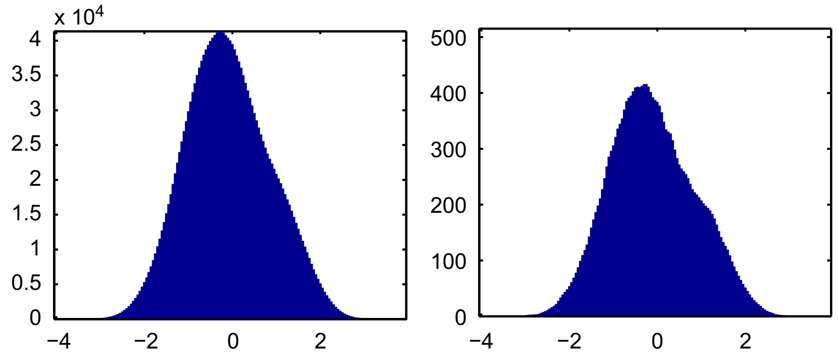

Fig. 1. Histograms of projection of dataset [31] and query set to a random direction. The vertical ordinate is the number of points and the horizontal ordinate is the projection value. The left is the histogram of dataset and the right is the histogram of query set. As shown in this figure, the projection of dataset and query set are both distributed in a small interval.

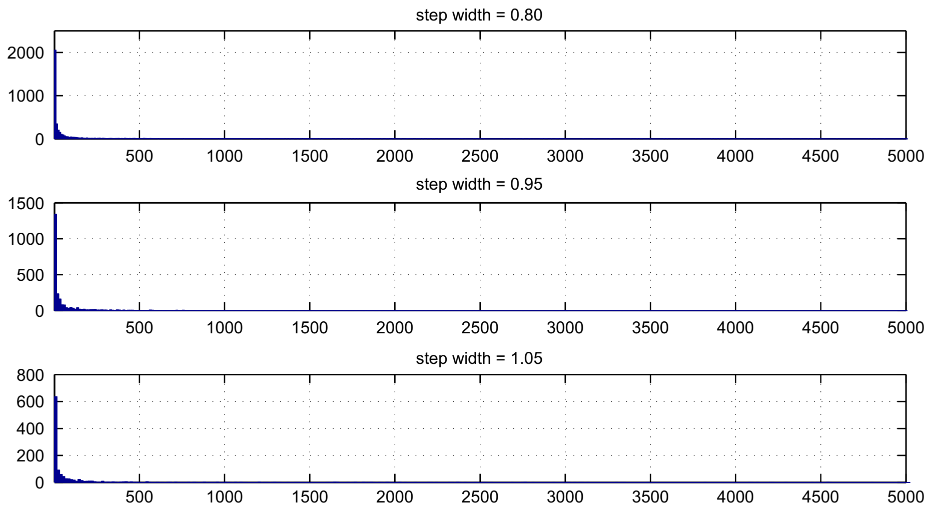

Fig. 2. Histograms of the number of data points in each bucket. The vertical ordinate is the number of buckets and the horizontal ordinate is the number of data points in each bucket. The step width W in LSH hash function is 0.80 for the top, 0.95 for the middle and 1.05 for the bottom. As shown in this figure, a little fraction of buckets contain a large part of data points, while most buckets contain a few data points.

will be more and more data points in a little fraction of buckets and the problem of "Non-Uniform" will be more and more serious. Consequently, the final filtering process based on exact similarity measure will be exhaustive for most queries and dramatically degrade the query efficiency.

The problem analysed above is intrinsic and inevitable for Euclidean LSH. To make Euclidean LSH scale to large-scale indexing, accelerating the final filtering process is essential.

2.2. *Using triangle inequality to prune the search*

Similarity search problem in specific metric space, e.g. Euclidean space, can be accelerated by using triangle inequality [32]. Burkhard–Keller Tree (BKT) [33], probably the first general solution of search problem in metric space, is designed for discrete distance functions and defined as follows. An arbitrary point $p \in U$ in dataset is selected as root of the tree. For each distance $i > 0$, define the set of all points at distance i to p as $U_i = \{u \in U, d(u,p) = i\}$. Then for each nonempty U_i, build a subtree of p and recursively build BKT for U_i (labelled i). This process is repeated recursively. Points selected as roots of all subtrees are called pivots. When given a query q and distance r, search process begins at the root and enter into all subtree i such that $d(p,q) - r \le i \le d(p,q) + r$, and proceed recursively. Since only a subset of U is searched, the query process is accelerated. A further development over BKT is fixed queries tree (FQT) [34], its structure is similar with BKT while all pivots in the nodes of same level are the same, and the actual data is stored at leaves. The advantage of such construction is that some comparisons between query and pivots are saved along the backtracking in the tree. A variant of FQT called "fixed-height FQT" (FHQT) is proposed in [35], where all leaves are at the same depth. In [36], fixed queries array (FQA) is presented. FQA can improve the search efficiency by using many more pivots than the original FHQT with the same memory.

The methods described above are designed for discrete distance functions, which are rare in practical applications. Thus many indexes for continuous distance functions are proposed. The "metric tree" presented in [37] is a tree data structure designed for continuous distance functions, and Yianilos and Chiueh [25,26] based on the same idea call it "vantage-point tree" or VPTs. VPTs are a kind of binary tree and built recursively as follows. An arbitrary point $p \in U$ is selected as the root of the tree and the median value of the set of all distances, $M = median\{d(p,u) : u \in U\}$, is calculated. Those points u such that $d(p,u) < M$ are stored in the left subtree, while those such that $d(p,u) > M$ are stored in the right subtree. To solve a query q with distance r, first measure $d = d(q,p)$, then if $d - r < M$ we enter into the left subtree, and if $d + r > M$ we enter into the right subtree. An extension of VPTs is "multi-vantage-point tree" (MVPT) [27,28]. Different with VPTs, MVPT uses $m-1$ uniform percentiles instead of just the median to build the index. In [38], "bisector trees" (BSTs) are proposed. BSTs are also a binary tree and built recursively as follows. In each node, two "centers" c_1 and c_2 are selected. Points closer to c_1 than c_2 are stored into the left subtree and those closer to c_2 are stored in the right subtree. For each of the two centers, its "covering radius", which is the maximum distance from the center to any other point in its subtree, is stored. When doing query, we enter into the subtree if $d(q,c_i) - r$ is not larger than the covering radius of c_i. Proposed in [37], the "generalized-hyperplane tree" (GHT) is identical to BSTs on construction while using the hyperplane between c_1 and c_2 instead of the covering radius as the pruning criterion. GHT is extended in [27] to an m-ary tree, called GNAT (geometric near-neighbor access tree), keeping the same essential idea while using m centers in each node. In [39] the M-tree (MT) data structure is presented. It has dynamic capabilities and good I/O performance. The structure of MT has some resemblances to GNAT: at each node, a set of representatives are chosen and the points that are closer to one representative are stored in the subtree rooted by that representative. When query, in each node, is compared to all representatives of the node and the search algorithm uses covering radius as criterion to pruning the search process.

All the methods described above, whether designed for discrete distance functions or continuous distance functions, use triangle inequality to prune the query process. As the similarity measure in Euclidean LSH is Euclidean distance, we can also use triangle inequality to accelerate the filtering process. The triangle inequality in Euclidean space is

$$\forall \boldsymbol{p},\boldsymbol{q},\boldsymbol{r} \in \mathbb{E}^n, |d(\boldsymbol{q},\boldsymbol{r}) - d(\boldsymbol{p},\boldsymbol{r})| \le d(\boldsymbol{q},\boldsymbol{p}) \tag{2}$$

Thus for each bucket, we can choose a point $\boldsymbol{r}$ ($\boldsymbol{r}$ can be a data point in this bucket or not) as pivot and precompute the distance between each data point $\boldsymbol{p}$ and $\boldsymbol{r}$. For a query $\boldsymbol{q}$, we compute $d(\boldsymbol{q},\boldsymbol{r})$ at the beginning, and when determining whether a data point $\boldsymbol{p}$ is similar with $\boldsymbol{q}$, we first compute $d' = |d(\boldsymbol{q},\boldsymbol{r}) - d(\boldsymbol{p},\boldsymbol{r})|$: if d' exceeds predefined similarity threshold, then it is not necessary to compute the exact distance $d(\boldsymbol{p},\boldsymbol{q})$. With different pivot $\boldsymbol{r}$, the speedup can be very different [32]. As introduced above, in [25–28], some base points, which are similar to our pivots, have been used to construct index structure to accelerate search process. However, these base points are all selected randomly. Our experiments in Section 4.1 show that random selection is not very effective.

3. Pivot-based filtering algorithm

In this section, we present a method to get an optimal pivot and propose our pivot-based algorithm.

3.1. *Data-based pivot point*

As we use triangle inequality to prune the exhaustive exact distance computation, the bigger the $d' = |d(\boldsymbol{q},\boldsymbol{r}) - d(\boldsymbol{p},\boldsymbol{r})|$ is, the higher the probability of d' exceeding the similarity threshold would be. Thus a proper criterion for pivot selection in d' should be as large as possible. To make d' large, pivot $\boldsymbol{r}$ should be close to the line through $\boldsymbol{p}$ and $\boldsymbol{q}$ and far away from $\boldsymbol{p}$ and $\boldsymbol{q}$ (if $\boldsymbol{r}$ is not in the line through $\boldsymbol{p}$ and $\boldsymbol{q}$). We give a simple proof as follows:

$$d(\boldsymbol{r},\boldsymbol{p_1})-d(\boldsymbol{r},\boldsymbol{p_2}) = d\sin\gamma/\sin\alpha - d\sin\beta/\sin\alpha \tag{3}$$

$$\begin{aligned} &= 2d\left(\cos\frac{\gamma+\beta}{2}\sin\frac{\gamma-\beta}{2}\right)\Big/\sin\alpha \\ &= 2d\sin\frac{\alpha}{2}\cos\left(\frac{\alpha}{2}+\beta\right)\Big/\sin\alpha \\ &= d\cos\left(\frac{\alpha}{2}+\beta\right)\Big/\cos\frac{\alpha}{2} \\ &= d\left(\cos\beta - \tan\frac{\alpha}{2}\sin\beta\right) \end{aligned} \tag{4}$$

As shown in the left of Fig. 3, $\boldsymbol{r}$ is the pivot point, $\boldsymbol{p_1}$ and $\boldsymbol{p_2}$ are two arbitrary points. $\boldsymbol{r}$, $\boldsymbol{p_1}$ and $\boldsymbol{p_2}$ make a triangle. $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and γ are three internal angles of the triangle. $\boldsymbol{d}$ is the distance of $\boldsymbol{p_1}$ and $\boldsymbol{p_2}$. Without loss of generality, we assume $d(\boldsymbol{r},\boldsymbol{p_1}) > d(\boldsymbol{r},\boldsymbol{p_2})$, thus we have $\gamma > \beta$ and $\gamma = \pi-\alpha-\beta$. From Eq. (4), it is easy to see that the smaller the α and β, the larger the $d(\boldsymbol{r},\boldsymbol{p_1})-d(\boldsymbol{r},\boldsymbol{p_2})$, thus $\boldsymbol{r}$ is closer to the line through $\boldsymbol{p_1}$ and $\boldsymbol{p_2}$ and farther away from $\boldsymbol{p_1}$ and $\boldsymbol{p_2}$.

Let $\boldsymbol{X} \subset R^d$ be the dataset in a bucket and query $\boldsymbol{q}$ is extracted from the distribution of $\boldsymbol{X}$, the quality of a pivot $\boldsymbol{r}$ can be measured by the Mean of Differences (MDs) between the distance of each data point $\boldsymbol{p}$ to $\boldsymbol{r}$ and the distance of query $\boldsymbol{q}$ to $\boldsymbol{r}$:

$$\mathrm{MD}(\boldsymbol{r}) = E(|d(\boldsymbol{q},\boldsymbol{r})-d(\boldsymbol{p},\boldsymbol{r})|) \tag{5}$$

$$\mathrm{MD}(\boldsymbol{r}) = \iint_{\boldsymbol{q},\boldsymbol{p}\in\boldsymbol{X}} p(\boldsymbol{q},\boldsymbol{p})|d(\boldsymbol{q},\boldsymbol{r})-d(\boldsymbol{p},\boldsymbol{r})|\,d\boldsymbol{q}\,d\boldsymbol{p} \tag{6}$$

where $\boldsymbol{q}$, $\boldsymbol{p} \in \boldsymbol{X}$ and $p(\boldsymbol{q},\boldsymbol{p})$ is the probability distribution function of $(\boldsymbol{q},\boldsymbol{p})$. Thus a pivot $\boldsymbol{r}$ can be obtained by solving the following optimization problem:

$$\boldsymbol{r} = \arg\max_{\boldsymbol{r}\in R^d} \mathrm{MD}(\boldsymbol{r}) = \arg\max_{\boldsymbol{r}\in R^d} E(|d(\boldsymbol{x_1},\boldsymbol{r})-d(\boldsymbol{x_2},\boldsymbol{r})|) \tag{7}$$

where $\boldsymbol{x_1}$, $\boldsymbol{x_2} \in \boldsymbol{X}$.

The middle of Fig. 3 illustrates how to choose a good pivot (red points): based on Eq. (4) and considering the data distribution, if pivot $\boldsymbol{r}$ is chosen in line with direction u_1, in which direction the data points distribute in a larger interval, $\mathrm{MD}(\boldsymbol{r})$ is larger than the one computed in the direction u_2 or u_3. We take the direction of the line through pivot $\boldsymbol{r}$ and mean of $\boldsymbol{X}$ (denoted by $\overline{\boldsymbol{x}}$) as pivot direction, with direction vector $\boldsymbol{\omega}$, thus pivot $\boldsymbol{r}$ can be set to

$$\boldsymbol{r} = \overline{\boldsymbol{x}} + L*\boldsymbol{\omega} \tag{8}$$

where L is the distance between $\overline{\boldsymbol{x}}$ and $\boldsymbol{r}$.

It can be seen from Eq. (8), to get $\boldsymbol{r}$ is to get $\boldsymbol{\omega}$ and L. We can shift pivot $\boldsymbol{r}$ to the origin. This operation does not change the value of Eq. (7). Let $\boldsymbol{X}^* = \{\boldsymbol{x}^* = \boldsymbol{x}-\boldsymbol{r} : \boldsymbol{x} \in \boldsymbol{X}\}$ be the shifted dataset and the optimization problem (7) can be rewritten as:

$$(\boldsymbol{\omega},L) = \arg\max_{(\boldsymbol{\omega}\in R^d, L\in R)} E(|\|\boldsymbol{x_1^*}\|_2 - \|\boldsymbol{x_2^*}\|_2|) \tag{9}$$

where $\boldsymbol{x_1^*}$, $\boldsymbol{x_2^*} \in \boldsymbol{X}^*$.

Since these data points are in the same bucket, the distribution scale in some direction is relatively small (otherwise the hash values of these data points are not same). If $\boldsymbol{r}$ is far from these data points (which means L is large enough), as shown in the right of Fig. 3, we would have $\theta_1,\theta_2 \to 0$, then we can make an approximation of $\|\boldsymbol{x_1^*}\|_2 - \|\boldsymbol{x_2^*}\|_2$:

$$\begin{aligned} |\|\boldsymbol{x_1^*}\|_2 - \|\boldsymbol{x_2^*}\|| &= \left|\frac{P_1\cos\theta_2 - P_2\cos\theta_1}{\cos\theta_1\cos\theta_2}\right| \\ &\approx \left|\frac{P_1-P_2}{\cos\theta_1 * \cos\theta_2} - o(P_1-P_2)\right| \overset{\theta_1,\theta_2\to 0}{\longrightarrow} |P_1-P_2| \end{aligned} \tag{10}$$

where P_1, P_2 are the projections of $\boldsymbol{x_1^*}$, $\boldsymbol{x_2^*}$ on $\boldsymbol{\omega}$.

Eq. (10) means that $\|\boldsymbol{x_1^*}\|_2 - \|\boldsymbol{x_2^*}\|_2$ is proportional to the difference between their projections on $\boldsymbol{\omega}$ when pivot $\boldsymbol{r}$ is far away. Thus, to maximize $\mathrm{MD}(\boldsymbol{r})$ (which is $E(|\|\boldsymbol{x_1^*}\|_2 - \|\boldsymbol{x_2^*}\|_2|)$ in Eq. (9)) is to maximize $E(|\boldsymbol{x_1^*}\cdot\boldsymbol{\omega} - \boldsymbol{x_2^*}\cdot\boldsymbol{\omega}|)$ when pivot $\boldsymbol{r}$ is far away. After being shifted to the origin, the absolute position of $\boldsymbol{r}$ is no longer needed to be considered. Moreover, since we set pivot $\boldsymbol{r}$ far away from all data points to get an approximation (Eq. (10)), L in Eq. (8) is no longer needed to consider either. Thus the pivot direction $\boldsymbol{\omega}$ is the only factor now. The optimization problem (9) can be rewritten as:

$$\boldsymbol{\omega} = \arg\max_{\boldsymbol{\omega}\in R^d} E(|\boldsymbol{x_1^*}\cdot\boldsymbol{\omega} - \boldsymbol{x_2^*}\cdot\boldsymbol{\omega}|) \tag{11}$$

where $\boldsymbol{x_1^*}$, $\boldsymbol{x_2^*} \in \boldsymbol{X}^*$. From Cauchy–Schwarz inequality we have

$$E(|\boldsymbol{x_1^*}\cdot\boldsymbol{\omega} - \boldsymbol{x_2^*}\cdot\boldsymbol{\omega}|) \le (E(|\boldsymbol{x_1^*}\cdot\boldsymbol{\omega} - \boldsymbol{x_2^*}\cdot\boldsymbol{\omega}|^2))^{1/2} \tag{12}$$

Eq. (12) gives an upper bound of $\mathrm{MD}(\boldsymbol{r})$. Maximizing the upper bound would guarantee a high probability to maximize the related $\mathrm{MD}(\boldsymbol{r})$. Let $Y = \{y = \boldsymbol{x}\cdot\boldsymbol{\omega} : \boldsymbol{x} \in \boldsymbol{X}\}$ be the projection set of the original dataset $\boldsymbol{X}$ to $\boldsymbol{\omega}$ and $Y^* = \{y^* = \boldsymbol{x}^*\cdot\boldsymbol{\omega} : \boldsymbol{x}^* \in \boldsymbol{X}^*\}$ be the projection set of the shifted dataset $\boldsymbol{X}^*$ to $\boldsymbol{\omega}$. It is easy to prove

$$E(|\boldsymbol{x_1^*}\cdot\boldsymbol{\omega} - \boldsymbol{x_2^*}\cdot\boldsymbol{\omega}|^2) = E(|y_1^* - y_2^*|^2) = 2\,\mathrm{Var}(Y^*) \tag{13}$$

where y_1^*, $y_2^* \in Y^*$ and $\mathrm{Var}(Y^*)$ is the variance of Y^*. Since the original dataset are shifted by subtracting $\boldsymbol{r}$, the change of all original projections is the same, thus

$$\mathrm{Var}(Y^*) = \mathrm{Var}(Y) = \mathrm{Var}(\boldsymbol{x}\cdot\boldsymbol{\omega}) \tag{14}$$

where $\boldsymbol{x} \in \boldsymbol{X}$.

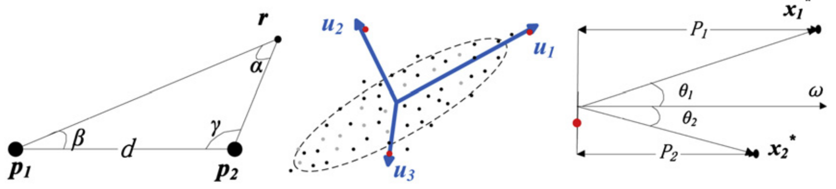

Fig. 3. Illustrations for the derivation of optimal pivot selection. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this article.)

From the above derivation, we can conclude that, to get an optimal pivot $\boldsymbol{r}$ is to get the pivot direction $\boldsymbol{\omega}$ and L in Eq. (8). When L is large enough, L no longer needs to be considered. Furthermore, pivot direction $\boldsymbol{\omega}$ can be obtained by maximizing $\mathrm{Var}(\boldsymbol{x}\cdot\boldsymbol{\omega})$ $(\boldsymbol{x}\in\boldsymbol{X})$. From principal component analysis [40], we know that when data points are projected to the eigenvector of covariance matrix with the largest eigenvalue, the variance of the distribution of projections would be largest. Thus $\boldsymbol{\omega}$ can be obtained by solving the following equation:

$$S\boldsymbol{\omega}=\lambda\boldsymbol{\omega} \tag{15}$$

where S is the covariance matrix of $\boldsymbol{X}$. The direction vector of eigenvector of S with the largest eigenvalue is selected as $\boldsymbol{\omega}$. So far, according to Eq. (8), we can get an optimal pivot $\boldsymbol{r}$.

3.2. Pivot-based filtering algorithm

After constructing LSH index, for each bucket, **UpdateIndex** is invoked to get m pivots $\boldsymbol{r_i}$ (reasons for using multi pivots is discussed in Section 3.3) and compute the distance between each pivot $\boldsymbol{r_i}$ and each data point in this bucket. Algorithm **UpdateIndex** is illustrated in Algorithm 1. When a query $\boldsymbol{q}$ is given, **QueryFiltering** is invoked to get result. It is illustrated in Algorithm 2. For range query, the initial similarity threshold T and the inner similarity thresholds t are set to the similar radius. For (Approximate) Nearest Neighbor query (ANN) and (Approximate) k-Nearest Neighbor query (k NN), the initial threshold T is set to ∞ and the inner similarity threshold t is variable during the query process: for ANN query, t is updated to the distance between $\boldsymbol{q}$ and the nearest neighbor encountered so far, and for k NN query t is updated to the distance between $\boldsymbol{q}$ and the k-th nearest neighbor encountered so far.

Algorithm 1. UpdateIndex

```
Input: a data bucket (with dataset $\boldsymbol{X}$ of size $N$) and pivot number $m$.
Output: an updated data bucket.
Initialization: set covariance matrix $S=0$, $\overline{\boldsymbol{x}}=0$;
for $i\leftarrow 1$ to $N$ do
  $S=S+\boldsymbol{x_i}\boldsymbol{x_i}^T$;
  $\overline{\boldsymbol{x}}=\overline{\boldsymbol{x}}+\boldsymbol{x_i}$;
end
$\overline{\boldsymbol{x}}=\overline{\boldsymbol{x}}/N$;
$S=S/N-\overline{\boldsymbol{x}}\overline{\boldsymbol{x}}^T$;
solve $S\omega=\lambda\omega$, get $m$ eigenvectors with the first $m$ largest
eigenvalue;
for $i\leftarrow 1$ to $m$ do
  set pivot $\boldsymbol{r_i}=\overline{\boldsymbol{x}}+L_i\boldsymbol{\omega_i}$;
  save pivot $\boldsymbol{r_i}$ to index;
end
for $i\leftarrow 1$ to $N$ do
  for $j\leftarrow 1$ to $m$ do
    calculate $d(\boldsymbol{x_i},\boldsymbol{r_j})$ and save it to index;
  end
end
```

Algorithm 2. QueryFiltering

```
Input: query $\boldsymbol{q}$, predefined similarity threshold $T$.
Output: query result List
Initialization: set inner similarity threshold $t=T$ and result List
empty;
calculate the hash value of $\boldsymbol{q}$ and enter into the corresponding
bucket, the bucket size is $N$ and the pivots in it is $\boldsymbol{r_i}$, $i\in[1,m]$;
for $i\leftarrow 1$ to $m$ do
  calculate $d(\boldsymbol{q},\boldsymbol{r_i})$;
end
for $i\leftarrow 1$ to $N$ do
  if there is no $j\in[1,m]$ such that $|d(\boldsymbol{q},\boldsymbol{r_j})-d(\boldsymbol{x_i},\boldsymbol{r_j})|>t$ then
    calculate $d(\boldsymbol{q},\boldsymbol{x_i})$;
    if $d(\boldsymbol{q},\boldsymbol{x_i})<t$ then
      update List with $\boldsymbol{x_i}$;
      update inner threshold $t$;
    end
  end
end
```

In Section 4, we conduct a series of experiments to test our algorithm, and the experimental results validate the effectiveness of our algorithm.

3.3. Algorithm analysis

As shown in our experiments in Section 4, our algorithm can accelerate the online query process of Euclidean LSH. However, there are still some issues needed to be discussed. In this section we analyse the factors that may affect the performance of our algorithm.

For online queries, since we make an assumption for obtaining an optimal pivot $\boldsymbol{r}$ in Section 3.1: $\boldsymbol{r}$ is far away from all data points in a bucket. The distance between pivot $\boldsymbol{r}$ and the mean vector of the dataset, which is L in Eq. (8), is a key factor. In Section 4.2, our experiment shows that the larger the L is, the better the speedup would be. However, when L exceeds a certain threshold, the speedup will be almost unchanged. This is reasonable and the reason is briefly discussed in Section 4.2.

Some metric space indexes [27,39] use more than one pivot to further prune the query process. We also use multipivots in our algorithm to get better speedup. Our experiments in Section 4.3 show it is feasible to use multipivots.

Another important factor is the memory occupation of our modified LSH index. Since the distance to pivot is saved, our modified LSH index needs more memory. For a standard LSH index, with N d-dimensional data points, k-dimensional hash value and L hash tables, the space complexity is at least $O(Nd+NkL)$. If we use one pivot, the space complexity is $O(Nd+NkL+NL+nL)$ (n is the average number of buckets in each hash table, so we have $n\ll N$), so the space complexity of our modified LSH index is $O(Nd+N(k+1+n/N)L)\approx O(Nd+N(k+1)L)$, which is similar with a LSH index with $k+1$ dimensional hash value.

The purpose of our algorithm is to accelerate the query process of Euclidean LSH. In theory, increasing the dimension of hash value will decrease the average number of data points in each bucket, thus the query process may be speeded up. It seems that increasing the dimension of hash value is a simple way to accelerate the query process while using the same memory as our algorithm. However, as the dimension of hash value increases, the probability of similar points hashed to the same bucket decreases, thus the quality of query degenerates. Moreover, for range query, the number of buckets needed to probe increases.

The time complexity of our algorithm for pivot selection is $O(Nd^2)$. When the algorithm is applied to large-scale and high-dimensional dataset, it is time-consuming. Even though this process is offline, we also want to accelerate it. From our derivation in Section 3.1, we can see that the data distribution in each bucket is a key factor for pivot selection. If we can sample a subset of representative data points, we may get a good enough estimation of pivot direction, thus accelerate the process of pivot selection. In Section 4.4, our experiments verify this idea.

4. Experiment

To evaluate the performance of the proposed method, we conduct experiments on two benchmark datasets: ANN_SIFT1M (1 million points) [31] and NUS-WIDE (270k points) [41]. ANN_SIFT1M dataset contains 1 million local SIFT descriptors extracted from random images. It also provides a query set which contains 10,000 data points. We directly use this query set as our query set. As the original SIFT points are not normalized and the norm of each data point is too large (almost 500), we normalize these data points by dividing each dimension by the largest norm in the dataset. NUS-WIDE dataset contains 269,684 images from Flickr. Six types of low-level features are extracted from these images. We use 144-D color correlogram to do our experiments. A test dataset which contains 107,859 feature points is also provided. We randomly select 10,000 feature points from the test set as our query set. For this dataset, we normalize all feature points by dividing each dimension by their l_2 norm.

Since each hash table in Euclidean LSH is independent, the effectiveness can be observed just by one hash table, thus in each experiment, we use one hash table and set the dimension of hash value $k=5$. The quantization step width W is set to 0.95. In the query process, the number of buckets probed is set to 20.

To evaluate the effectiveness of our method, for each query $\boldsymbol{q}$, we record the number of points skipped (the actual distance of these points to $\boldsymbol{q}$ is not calculated) during the query process. The final speedup is averaged among 10,000 queries. The acceleration of the proposed method for rang query, (approximate) nearest neighbor (ANN) query and k-nearest neighbor query (k NN) are evaluated respectively. To prove the advantage of our optimal pivot selection method which is our primary contribution, we compare it with the random selection method which is used in [25–28]. In addition, we demonstrate the influence of parameter selection through experiments. At last, we verify the effectiveness of sampling method to reduce the off-line computing cost. All the experiments are done on a server with a 64-bit 2.4 GHz Quadcore CPU and 24 GB RAM.

4.1. Experimental results

For a range query, we vary the similar radius and evaluate the speedup respectively. As shown in Figs. 4 and 5, with the proposed method, the filtering process is significantly accelerated, e.g. 220% for radius=0.3 and 140% for radius=0.6 on ANN_SIFT1M dataset. On NUS-WIDE dataset, the proposed method gets more obvious acceleration, which is 320% for radius=0.2 and 160% for radius=0.5. We can observe that, as the similar radius increases, the speedup decreases. This is reasonable, because for a query $\boldsymbol{q}$ and a data point $\boldsymbol{p}$, $|d(\boldsymbol{q},\boldsymbol{r})-d(\boldsymbol{p},\boldsymbol{r})|$ has a maximum $d(\boldsymbol{q},\boldsymbol{p})$, and when similar radius increases, the probability of $|d(\boldsymbol{q},\boldsymbol{r})-d(\boldsymbol{p},\boldsymbol{r})|$ larger than similar radius decreases, thus the speedup decreases. When the similar radius is larger than $d(\boldsymbol{q},\boldsymbol{p})$, no matter where pivot $\boldsymbol{r}$ is, $|d(\boldsymbol{q},\boldsymbol{r})-d(\boldsymbol{p},\boldsymbol{r})|$ is always smaller than the similar radius and the computation of $d(\boldsymbol{q},\boldsymbol{p})$ is inevitable. We also

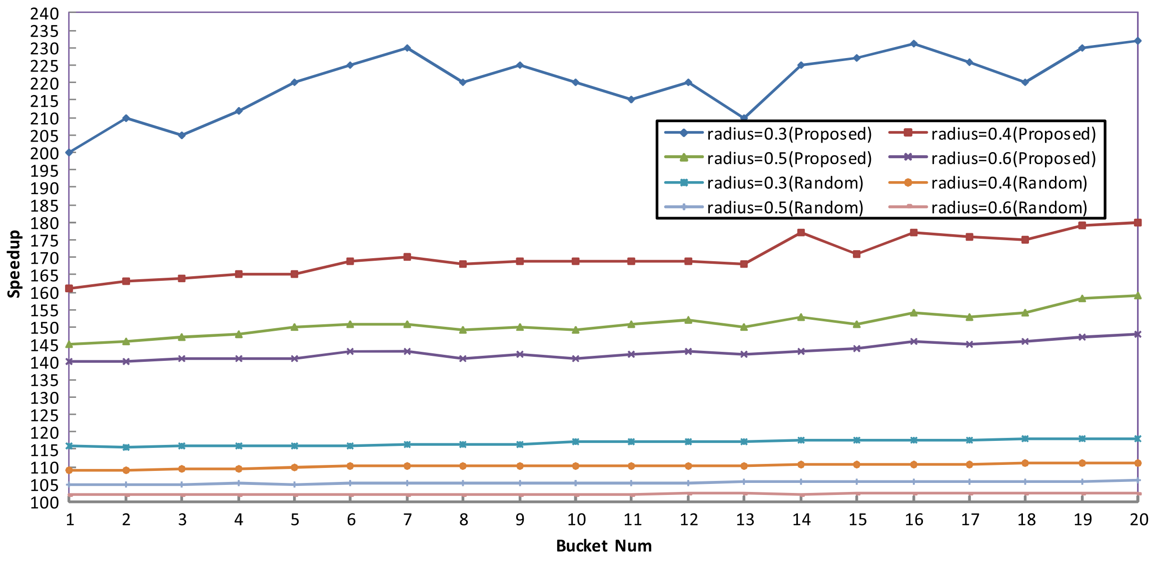

Fig. 4. Speedup for range query on ANN_SIFT1M dataset using our data-based pivot and randomly selected pivot under different similar radii. The horizontal ordinate is number of buckets probed and the vertical ordinate is percentage of speedup. As shown in this figure, the proposed method is effective for range query and outperforms the random selection method.

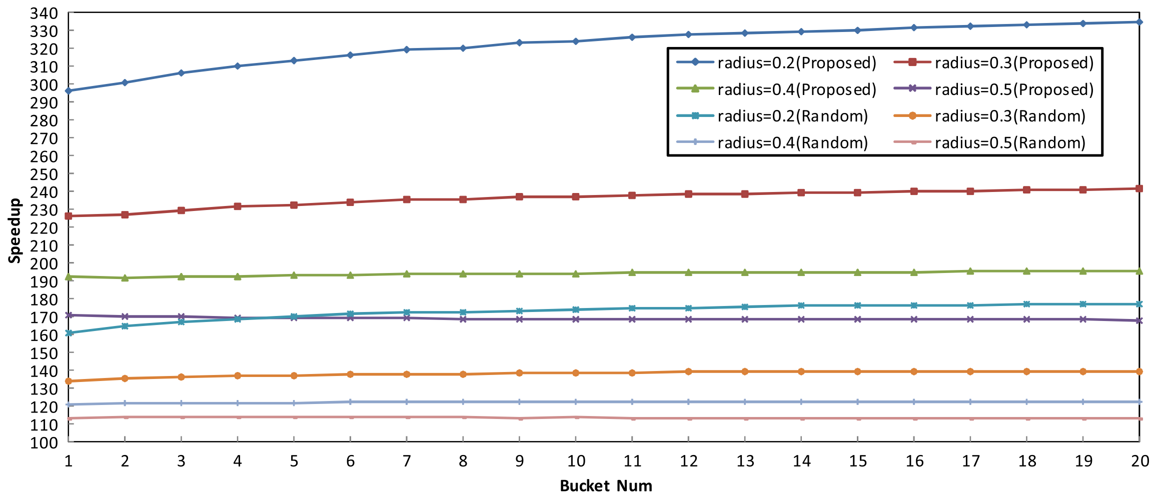

Fig. 5. Speedup for range query on NUS-WIDE dataset using our data-based pivot and randomly selected pivot under different similar radii. The horizontal ordinate is number of buckets probed and the vertical ordinate is percentage of speedup. As shown in this figure, the proposed method is effective for range query and outperforms the random selection method.

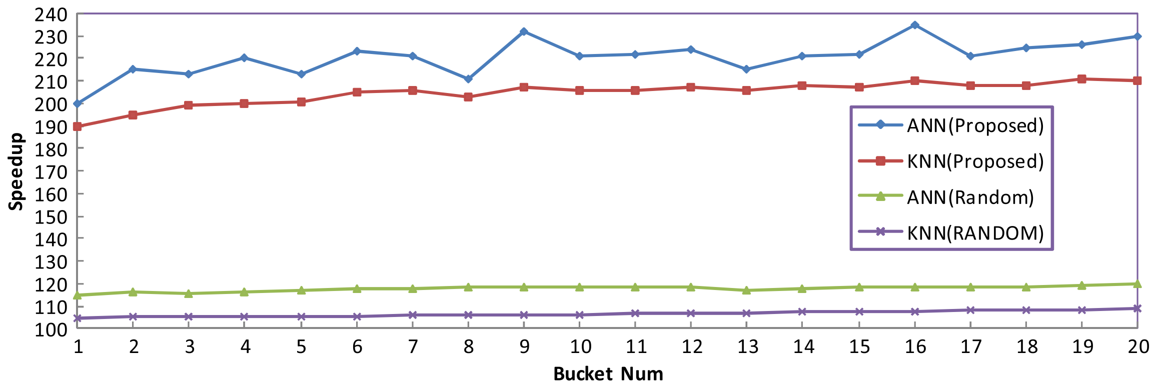

Fig. 6. Speedup for ANN and k NN query on ANN_SIFT1M dataset using our data-based pivot and randomly selected pivot. The horizontal ordinate is the number of buckets probed and the vertical ordinate is percentage of speedup. By using our method, the speedup is almost 230% for ANN query and 200% for k NN query. However, the speedup of random selection method is obviously lower than the proposed method, which is 120% for ANN and 110% for KNN at most.

compare our data-based pivot with randomly selected pivot. As shown in Figs. 4 and 5, our data-based pivot outperforms randomly selected pivot. In fact, randomly selected pivot is not very effective.

Figs. 6 and 7 show the experimental results for ANN and k NN query on both datasets. For k NN query, we set $k=100$. The speedup of the proposed pivot point is almost 230% for ANN and 200% for k NN on ANN_SIFT1M dataset in Fig. 6, while the speedup of randomly selected pivot point is less than 120% for ANN and 110% for k NN. The speedup of the proposed method can reach 260% for ANN and 210% for KNN on NUS-WIDE dataset in Fig. 7. Although the speedup of randomly selected pivot point can reach 150% for ANN and 130% for KNN, it is significantly lower than the proposed method.

From the experimental results in this section, we can conclude that by using our algorithm, the query process of Euclidean LSH can be accelerated significantly. In addition, our method is used in reranking stage. Even if the index can be optimized by a careful implementation or parameter tuning to get a better query efficiency, by using our method, the query efficiency can be further improved. That is to say, our method can accelerate Euclidean LSH in any case.

4.2. Speedup with different L

In this section, we conduct three experiments to demonstrate the relationship between the speedup of our algorithm and L in Eq. (8). We increase L gradually and calculate the speedup. As shown in Figs. 8 and 9, the larger the L is, the better the speedup would be. However, when L is greater than 4.0 in our experiments, the speedup of our algorithm is almost unchanged. This is because there is a maximum of Eq. (3). When L is large enough, which means pivot $\boldsymbol{r}$ is far away enough, the difference of distances between pivot $\boldsymbol{r}$ and any two data points is almost unchanged, thus the speedup is almost unchanged.

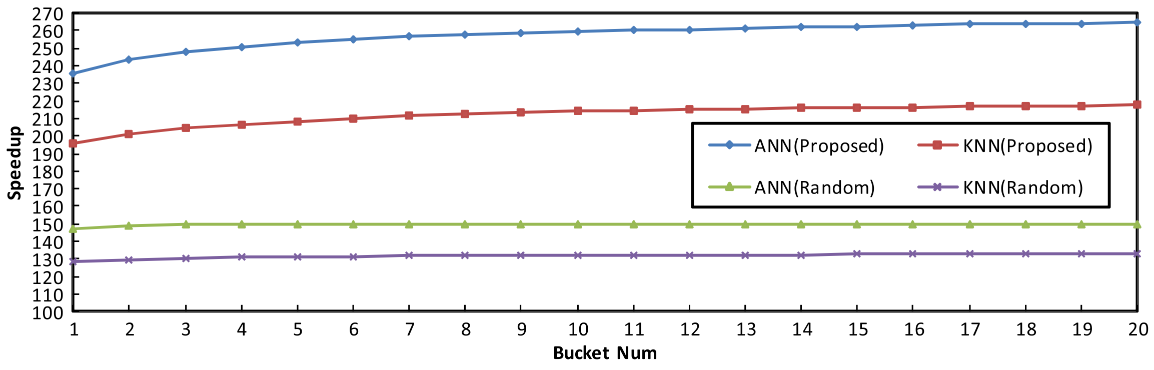

Fig. 7. Speedup for ANN and *k* NN query on NUS-WIDE dataset using our data-based pivot and randomly selected pivot. The horizontal ordinate is number of buckets probed and the vertical ordinate is percentage of speedup. By using our method, the speedup can reach 260% for ANN query and 210% for *k* NN query. However, the speedup of random selection method is significantly lower than the proposed method, which is 150% for ANN query and 130% for KNN query at most.

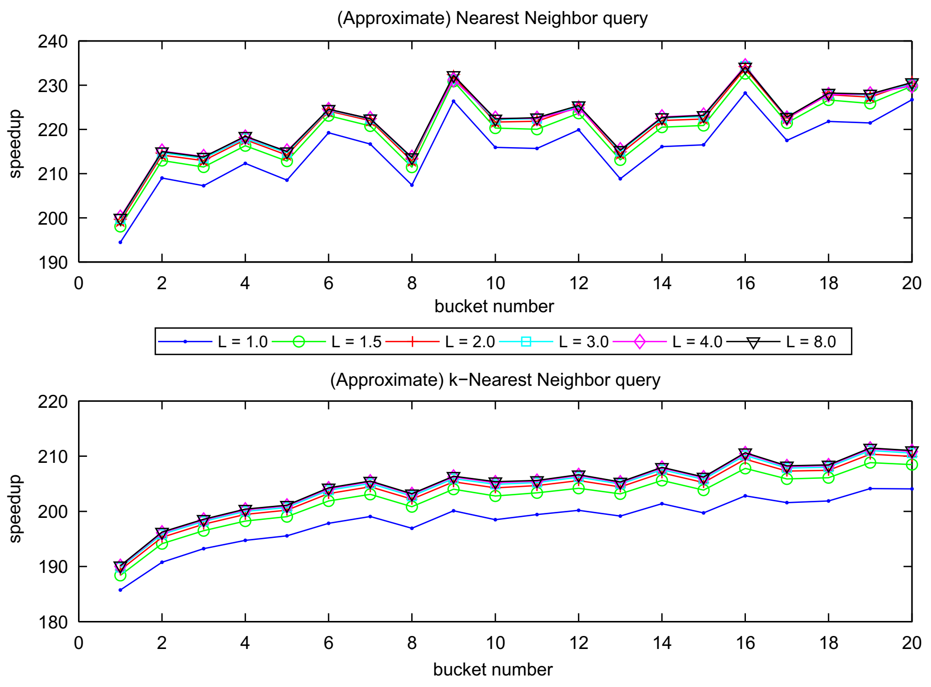

Fig. 8. Speedup obtained for ANN and *k* NN query under different *L*. The top is the speedup for ANN query and the bottom is for *k* NN query. As shown, the large the *L* is, the better the speedup would be. When *L* exceeds a certain threshold (4.0), the speedup is almost unchanged, and this is reasonable.

4.3. *Speedup with multipivots*

In our previous experiments, we have noticed that when a data bucket contains many data points, the second largest eigenvalue, sometimes even the third largest eigenvalue, is almost larger than 50% of the largest eigenvalue. This observation means that the distribution of data projection on the eigenvector with second largest, or even third largest, eigenvalue is also large. According to our derivation in Section 3.1, choosing a pivot in these directions may also bring considerable speedup. In the following experiment, we use two pivots and three pivots for range query respectively. As shown in Fig. 10, by using multipivots, the speedup of the query process is even higher. With two pivots, the speedup is almost 300% for range radius=0.3. With three pivots, the speedup is almost 400%.

4.4. *Sampling method*

The eigenvector of the covariance matrix of all data points in each bucket is necessary for our algorithm. When the data dimension is high and the number of points is large, the calculation of eigenvector is time-consuming. Even this process is offline, it is valuable to accelerate the process. In Eq. (8), a pivot in each bucket consists of two parts: mean vector of the data points in this bucket ($\overline{\boldsymbol{x}}$) and pivot direction vector ($\boldsymbol{\omega}$), which is the eigenvector of covariance matrix of all data points in this bucket. The calculation of $\overline{\boldsymbol{x}}$ is fast while the calculation of

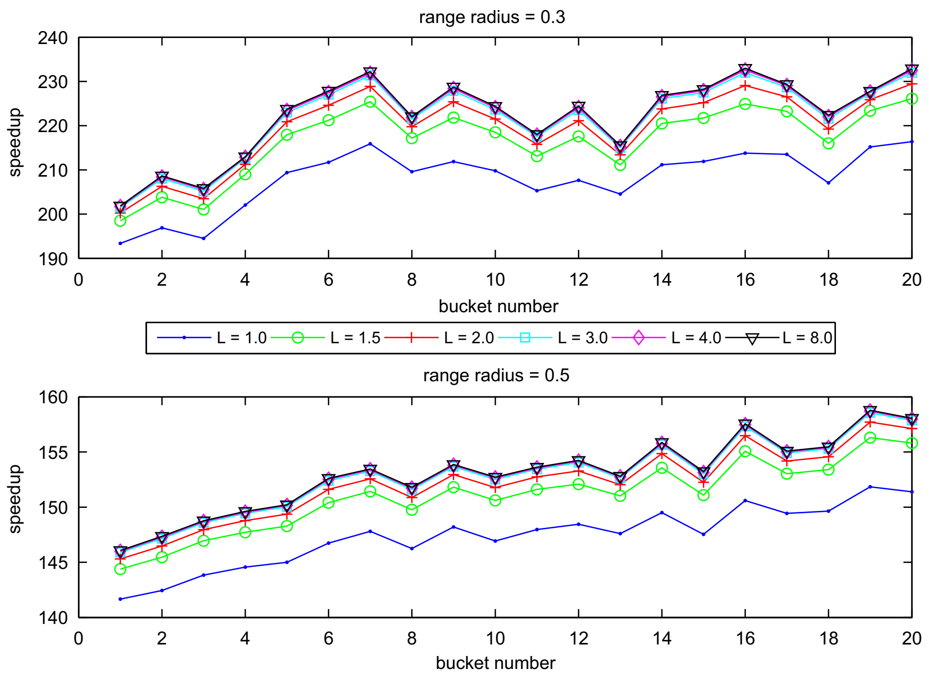

Fig. 9. Speedup obtained for range query under different L. Similar radius is 0.3 in the top and 0.5 in the bottom. We can also see that, the large the L is, the better the speedup would be. When L is larger than 4.0, the speedup is almost unchanged.

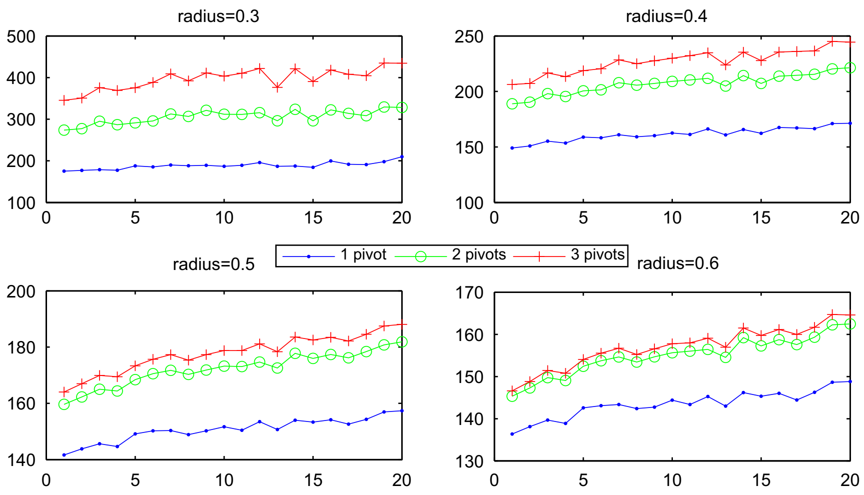

Fig. 10. Speedup obtained for range query under different number of pivots and different similar radii. As shown in this figure, with more pivots, the speedup is higher, and this is confirmed under different similar radii.

eigenvector ω is time-consuming. From our derivation in Section 3.1, we can see that there is a strong correlation between the data distribution and the eigenvector ω. A reasonable idea is that if we could sample a subset of representatives from all data points, we may get a good enough estimation of data distribution and get a good enough estimation of eigenvector ω. Thus the process of pivot selection is accelerated.

We conduct an experiment to test our idea. The sample set used by us is the learning set of ANN_SIFT1M dataset [31]. First we build a LSH index with the sample set and calculate the pivot direction ω_{smp} in each bucket, then we insert the reference dataset into this pre-built LSH index, and calculate the pivot direction ω_{all} after insertion. We calculate the angle between ω_{all} and ω_{smp} (denoted by $\theta(\omega_{all},\omega_{smp})$), the smaller the $\theta(\omega_{all},\omega_{smp})$ is, the better the pivot direction estimated from sample set would be. There are some buckets which the sample set may not be hashed to, thus we also record the number of buckets that the sample set is hashed to. In our derivation in Section 3.1, we set pivot in the line with direction of eigenvector and far away from the data points. Because of symmetry, the eigenvector has two directions which are opposite to each other, thus both ω_{smp} and $-\omega_{smp}$ can be

Table 1
Distribution of $\theta(\boldsymbol{\omega_{all}},\boldsymbol{\omega_{smp}})$ in each bucket.

Range of $\theta(\boldsymbol{\omega_{smp}},\boldsymbol{\omega_{all}})$	Percentage of buckets (%)
[0°,5°)	13.22
[5°,10°)	17.31
[10°,15°)	12.73
[15°,20°)	11.35
[20°,25°)	9.14
[23°,30°)	4.24
[30°,50°)	9.96
[50°,70°)	9.22
[70°,90°)	12.82

used as an estimation of $\boldsymbol{\omega_{all}}$ when L is large enough. It means if $\theta(\boldsymbol{\omega_{all}},\boldsymbol{\omega_{smp}})$ is larger than 90°, we can use $\theta(\boldsymbol{\omega_{all}},-\boldsymbol{\omega_{smp}})$ ($<90°$) to present the angle between $\boldsymbol{\omega_{all}}$ and $\boldsymbol{\omega_{smp}}$ in our calculation of distribution of $\theta(\boldsymbol{\omega_{all}},\boldsymbol{\omega_{smp}})$. For example, if $\theta(\boldsymbol{\omega_{all}},\boldsymbol{\omega_{smp}})=175°$, then we can consider the angle between $\boldsymbol{\omega_{all}}$ and $\boldsymbol{\omega_{smp}}$ as 5° in the calculation of distribution $\theta(\boldsymbol{\omega_{all}},\boldsymbol{\omega_{smp}})$ because the effectiveness of $\boldsymbol{\omega_{smp}}$ and $-\boldsymbol{\omega_{smp}}$ is same.

Table 1 shows the distribution of $\theta(\boldsymbol{\omega_{all}},\boldsymbol{\omega_{smp}})$. As shown, in almost 54.6% buckets, $\theta(\boldsymbol{\omega_{all}},\boldsymbol{\omega_{smp}})$ is smaller than 20° which means the estimation of pivot direction is good enough in a sufficient number of buckets. Moreover, the fraction of the buckets that the sample set is hashed to is almost 56.3% in average (the dimension of hash value k is 5 in our experiment). From these observations, we can conclude that, if a good enough sample set is provided, accelerating our algorithm with sampling method is feasible.

5. Conclusion

LSH is efficient to index high-dimensional data and its variations can make it index large-scale dataset, thus it has been a popular index structure for large-scale and high-dimensional dataset. However, a final filtering process based on exact similarity measure is inevitable in query process. In this paper, we analyse the phenomenon we call "Non-Uniform" that dramatically degrades the query performance of the most popular Euclidean LSH. "Non-Uniform" will make a large proportion of queries hashed to a little fraction of buckets that contain too many data points with high probability. It causes an exhaustive computation in the filtering process. Therefore, the query performance significantly degrades, especially when the dataset is large-scale. We propose a pivot-based algorithm to accelerate the filtering process and a method to get an optimal pivot. In addition, we analyse some factors that may affect the performance of our algorithm and propose a sampling method to make our algorithm more feasible. The experiments show that by using our algorithm, with little memory enlargement, the filtering process can be significantly accelerated. In addition, our method is used in reranking stage. Even if the index can be optimized by a careful implementation or parameter tuning to get a better query efficiency, by using our method, the query efficiency can be further improved. That is to say, our method can accelerate Euclidean LSH in any case. Although our algorithm is designed for Euclidean LSH, it can be easily transplanted into other index structures.

Acknowledgements

This work is supported by National Key Technology Research and Development Program of China (2012BAH39B02); Co-building Program of Beijing Municipal Education Commission.

References

[1] D. Gorisse, M. Cord, F. Precioso, Locality-sensitive hashing for chi2 distance, IEEE Transactions on Pattern Analysis and Machine Intelligence 34 (2012) 402–409.
[2] M. Wang, X.-S. Hua, R. Hong, J. Tang, G.-J. Qi, Y. Song, Unified video annotation via multigraph learning, IEEE Transactions on Circuits and Systems for Video Technology 19 (2009) 733–746.
[3] A. Gionis, P. Indyk, R. Motwani, Similarity search in high dimensions via hashing, in: Proceedings of 25th International Conference on Very Large Data Bases (VLDB), 1999, pp. 518–529.
[4] M. Datar, N. Immorlica, P. Indyk, V. Mirrokni, Locality-sensitive hashing scheme based on p-stable distributions, in: Proceedings of the Twentieth Annual Symposium on Computational Geometry, SCG '04, ACM, New York, NY, USA, 2004, pp. 253–262.
[5] A. Andoni, P. Indyk, E2LSH: Exact Euclidean Locality Sensitive Hashing, 2004. URL 〈http://web.mit.edu/andoni/www/LSH/〉.
[6] A. Andoni, P. Indyk, Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions, Communications of the ACM 51 (2008) 117–122.
[7] B. Kulis, K. Grauman, Kernelized locality-sensitive hashing for scalable image search, in: Proceedings of 12th International Conference on Computer Vision, 2009, pp. 2130–2137.
[8] Y. Ke, R. Sukthankar, L. Huston, Efficient near-duplicate detection and sub-image retrieval, in: ACM Conference on Multimedia, 2004, pp. 869–876.
[9] G. Shakhnarovich, T. Darrell, P. Indyk, Nearest-Neighbor Methods in Learning and Vision: Theory and Practice, MIT Press, 2006.
[10] B. Matei, Y. Shan, H. Sawhney, Y. Tan, R. Kumar, D. Huber, M. Hebert, Rapid object indexing using locality sensitive hashing and joint 3D-signature space estimation, IEEE Transactions on Pattern Analysis and Machine Intelligence 28 (7) (2006) 1111–1126.
[11] M. Wang, X.-S. Hua, J. Tang, R. Hong, Beyond distance measurement: constructing neighborhood similarity for video annotation, IEEE Transactions on Multimedia 11 (2009) 465–476.
[12] M. Wang, X.-S. Hua, T. Mei, R. Hong, G.-J. Qi, Y. Song, L.-R. Dai, Semi-supervised kernel density estimation for video annotation, Computer Vision and Image Understanding 113 (3) (2009) 384–396.
[13] M. Wang, K. Yang, X.-S. Hua, H.-J. Zhang, Towards a relevant and diverse search of social images, IEEE Transactions on Multimedia 12 (8) (2010) 829–842.
[14] R. Hong, M. Wang, M. Xu, S. Yan, T.-S. Chua, Dynamic captioning video accessibility enhancement for hearing impairment, in: ACM International Conference on Multimedia (ACM MM), 2010.
[15] P. Indyk, N. Thaper, Fast image retrieval via embeddings, in: Proceedings of the International Workshop on Statistical and Computational Theories of Vision, 2003.
[16] Rina Panigrahy, Entropy based nearest neighbor search in high dimensions, in: Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm, SODA '06, ACM, New York, NY, USA, 2006, pp. 1186–1195.
[17] Q. Lv, W. Josephson, Z. Wang, M. Charikar, K. Li, Multi-probe LSH: efficient indexing for high-dimensional similarity search, in: Proceedings of the 33rd International Conference on Very Large Data Bases, VLDB '07, VLDB Endowment, 2007, pp. 950–961.
[18] A. Joly, O. Buisson, A posteriori multi-probe locality sensitive hashing, in: Proceedings of the 16th ACM International Conference on Multimedia, MM '08, ACM, New York, NY, USA, 2008, pp. 209–218.
[19] H. Jegou, L. Amsaleg, C. Schmid, P. Gros, Query-adaptative locality sensitive hashing, in: International Conference on Acoustics, Speech, and Signal Processing, 2008.

[20] Y. Weiss, A. Torralba, R. Fergus, Spectral hashing, The Neural Information Processing Systems 21 (2008) 1753–1760.
[21] J. Wang, S. Kumar, S.-F. Chang, Sequential projection learning for hashing with compact codes, in: Proceedings of the 27th International Conference on Machine Learning, 2010.
[22] W. Liu, J. Wang, S. Kumar, S.-F. Chang, Hashing with graphs, in: Proceedings of the 28th International Conference on Machine Learning, 2011.
[23] J. Wang, S. Kumar, S.-F. Chang, Semi-supervised hashing for scalable image retrieval, in: CVPR, 2010.
[24] M. Bawa, T. Condie, P. Ganesan, LSH forest: self-tuning indexes for similarity search, in: WWW 2005, 2005, pp. 651–660.
[25] P.N. Yianilos, Data structures and algorithms for nearest neighbor search in general metric spaces, in: Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '93, 1993, pp. 311–321.
[26] T. Chiueh, Content-based image indexing, in: Proceedings of the 20th International Conference on Very Large Data Bases, VLDB '94, 1994, pp. 582–593.
[27] S. Brin, Near neighbor search in large metric spaces, in: Proceedings of the 21st International Conference on Very Large Data Bases, VLDB '95, 1995, pp. 574–584.
[28] T. Bozkaya, M. Ozsoyoglu, Distance-based indexing for high-dimensional metric spaces, in: ACM SIGMOD Record, vol. 26, ACM, 1997, pp. 357–368.
[29] L. Zhang, X. Gu, Y. Zhang, D. Zhang, J. Li, A pivot-based filtering algorithm for enhancing query performance of LSH, in: Visual Communications and Image Processing (VCIP), 2011, pp. 1–4.
[30] P. Indyk, Stable distributions, pseudorandom generators, embeddings and data stream computation, in: FOCS, Published by the IEEE Computer Society, 2000, p. 189.
[31] H. Jgou, M. Douze, C. Schmid, Product quantization for nearest neighbor search, IEEE Transactions on PAMI 33 (1) (2011) 117–128.
[32] E. Chávez, G. Navarro, R. Baeza-Yates, J.L. Marroquín, Searching in metric spaces, ACM Computing Surveys 33 (2001) 273–321.
[33] W.A. Burkhard, R.M. Keller, Some approaches to best-match file searching, Communications of the ACM 16 (1973) 230–236.
[34] R. Baeza-Yates, W. Cunto, U. Manber, S. Wu, Proximity matching using fixed-queries trees, in: Combinatorial Pattern Matching, Springer, 1994, pp. 198–212.
[35] R. Baeza-Yates, Searching: an algorithmic tour, Encyclopedia of Computer Science and Technology 37 (1997) 331–359.
[36] E. Chávez, J. Marroquín, G. Navarro, Fixed queries array: a fast and economical data structure for proximity searching, Multimedia Tools and Applications 14 (2) (2001) 113–135.
[37] J. Uhlmann, Satisfying general proximity/similarity queries with metric trees, Information Processing Letters 40 (4) (1991) 175–179.
[38] I. Kalantari, G. McDonald, A data structure and an algorithm for the nearest point problem, IEEE Transactions on Software Engineering (1983) 631–634.
[39] P. Ciaccia, M. Patella, P. Zezula, M-tree: an efficient access method for similarity search in metric spaces, in: Proceedings of the 23rd International Conference on Very Large Data Bases, VLDB '97, 1997, pp. 426–435.
[40] I.T. Jolliffe, Principal component analysis, in: Springer Series in Statistics, 2nd edition, Springer, 2002.
[41] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, Y.-T. Zheng, Nus-wide: a real-world web image database from National University of Singapore, in: ACM International Conference on Image and Video Retrieval, 2009.