A Pivot-based Filtering Algorithm for Enhancing Query Performance of LSH

Lei Zhang[1,2], Xiao-guang Gu[1,2], Yong-dong Zhang[1], Dong-ming Zhang[1] and Jin-tao Li[1]
[1]Institute of Computing Technology, Chinese Academy of Sciences
[2]Graduate University of Chinese Academy of Sciences
Beijing, China
Email: {zhanglei09, xggu, zhyd, dmzhang, jtli}@ict.ac.cn

***Abstract*—In recent years, Locality Sensitive Hashing (LSH) (and its variant Euclidean LSH) has become a popular index structure for large-scale and high-dimensional similarity search problem. In this paper, we analyze a phenomenon we called "Non-Uniform" that degrades the query performance of LSH and propose a pivot-based algorithm to improve the query performance. We also provide a method to get optimal pivot for even larger improvement. Experiments show that our algorithm significantly improves the query performance of LSH.**

I. Introduction

As the problem of high-dimensional similarity search becomes more popular in content-based search systems, an efficient high-dimensional index is more desirable. Many index structures for approximate similarity search have been proposed. Among these, Locality Sensitive Hashing (LSH) [1], [2] is a popular high-dimensional index. The first successful variant of LSH, Euclidean LSH [3], expands the application range, but requires too many hash tables to guarantee the query accuracy. In order to reduce the memory consumption, other improved versions [4], [5], [6] have been proposed. All these variants of LSH are based on the same structure as Euclidean LSH.

LSH is efficient to organize and query large-scale and high-dimensional databases. However, when using LSH for query, a final filtering process based on exact similarity measure is needed. When the database is large-scale, which is common in practical applications, the number of points needed to filter is large; thus the cost of filtering is the domain factor that degrades the query performance. Euclidean LSH uses quantization of the projection of a data point to a randomly selected direction as the hash value, which makes the number of points in some buckets is significantly larger than others when constructing index; maps a query to a bucket containing too many data points with high probability when querying. This phenomenon, which we called "Non-Uniform", makes the cost of filtering process significantly higher, and moreover, when the dataset is large-scale, the problem of "Non-Uniform" will be even worse.

Our main contribution is to propose a pivot-based algorithm using Triangle Inequality to accelerate the filtering process of Euclidean LSH. In addition, we provide a method to get an optimal pivot. In Section II, we state a formal analysis of "Non-Uniform" of Euclidean LSH. In Section III, we introduce our algorithm and optimal pivot selection. The corresponding experiments are described in Section IV.

II. Problem Analysis

The basic idea of LSH is to use hash functions to hash similar points to same bucket with higher probability than dissimilar points. Let $\mathcal{S}$ be the domain of data point and $\mathcal{D}$ be the distance measure. A function family $\mathrm{H} = \{\mathrm{h}: \mathcal{S} \to \mathrm{U}\}$ is called $(\mathrm{r}_1, \mathrm{r}_2, \mathrm{p}_1, \mathrm{p}_2)$-sensitive for $\mathcal{D}$ if for any $p, q \in \mathcal{S}$:

$$\text{if } \mathcal{D}(p,q) \le \mathrm{r}_1 \text{, then } \Pr(\mathrm{h}(p) = \mathrm{h}(q)) \ge \mathrm{p}_1$$

$$\text{if } \mathcal{D}(p,q) \ge \mathrm{r}_2 \text{, then } \Pr(\mathrm{h}(p) = \mathrm{h}(q)) \le \mathrm{p}_2$$

where $\mathrm{p}_1 > \mathrm{p}_2$ and $\mathrm{r}_1 < \mathrm{r}_2$ to ensure the function family H is useful. Hash function used in LSH is defined as:

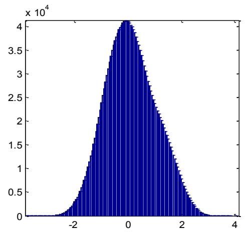
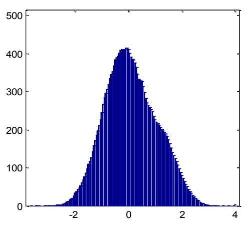
$$\mathrm{G} = \{\mathrm{g}: \mathcal{S} \to \mathrm{U}^{\mathrm{K}}\}$$

where $\mathrm{g}(p) = (\mathrm{h}_1(p), \mathrm{h}_2(p), \dots, \mathrm{h}_{\mathrm{K}}(p))$ and $\mathrm{h}_{\mathrm{i}} \in \mathrm{H}$. 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. The hash function used in Euclidean LSH is:

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

where W is a positive real number and b is chosen form uniform distribution $\mathrm{U}[0, W]$; $\boldsymbol{a}$ is a vector with the same dimension as $\boldsymbol{x}$ and each component is chosen independently from standard Gaussian distribution. Since Gaussian distribution is a 2-stable distribution, the distribution of $\boldsymbol{ax}/W$ is $\mathcal{N}(0, \|\boldsymbol{x}/W\|_2^2)$. Thus the probability of $\boldsymbol{ax}/W$ between $(-2\|\boldsymbol{x}/W\|_2, 2\|\boldsymbol{x}/W\|_2)$ is almost 95.5%. Moreover, in order to hash similar points to same bucket with high probability, W is always comparable to $\|\boldsymbol{x}\|$, which means $\|\boldsymbol{x}/\mathrm{W}\|$ is not very large. Therefore, a majority of $\boldsymbol{ax}/W$ is distributed in a small interval (shown in Fig. 1a), thus the hash value is distributed in a small interval. This phenomenon also occurs when query: the hash value of query

This work is supported by the National Basic Research Program of China (973Program, 2007CB311100); National High Technology and Research Development Program of China (863 Program, 2009AA01A403); Co-building Program of Beijing Municipal Education Commission.

(a) **(b)**

Fig. 1. Distribution histograms of projection of dataset [10] and query set. Vertical ordinate is the number of points and horizontal ordinate is the projection value. (a) histogram of the dataset, (b) histogram of the query set. As shown in this figure, the projection of all data points and query points are both distributed in a small interval.

point is also distributed in a small interval (shown in Fig. 1b), thus a query will be hashed to a bucket containing too many points with high probability.

From the above analysis, we can conclude that when the dataset is large-scale, there will be more data points in a bucket and the problem of "Non-Uniform" will be more serious. Therefore, the final filtering process based on exact similarity measure will be exhaustive and seriously degrade the query performance.

III. Pivot-based Filtering Algorithm

As the similarity measure is Euclidean distance in Euclidean LSH, we may use Triangle Inequality, which has been used in many Metric Space index structures [7], to accelerate the filtering process. Triangle Inequality in Euclidean space is:

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

Thus for each bucket, we choose a point as **Pivot Point** (denoted by ***PP*** in the following, the corresponding vector is denoted by ***Vpp***), precompute the distance between each data point and ***PP***, and add the distance to index. For a query $\boldsymbol{q}$, we compute $d(\boldsymbol{q},\boldsymbol{PP})$ at the beginning, and when determining whether a data point $\boldsymbol{p}$ is similar with $\boldsymbol{q}$, we could first compute $d' = |d(\boldsymbol{q},\boldsymbol{PP}) - d(\boldsymbol{p},\boldsymbol{PP})|$: if d' exceeds similarity threshold, then it is not necessary to compute the exact distance $d(\boldsymbol{p},\boldsymbol{q})$. With different ***PP***, the computation reduced is very different. In this section, we present a method to get an optimal ***PP*** and propose our pivot-based algorithm.

A. Randomly Selected Pivot Point

After constructing index, for each bucket, we randomly choose a data point as ***PP*** and compute the distance between other data points and ***PP***. The advantage of this method is: there is no effect on the dynamics of index. When adding or deleting data points, it is not necessary to alter ***PP*** and recompute all distances. It's a simple method while does not take account of the information provided by the data. In Section IV, our experiment shows that the query performance can be improved by this method, but not very impressive.

B. Data-based Pivot Point

As we use Triangle Inequality to avoid exhaustive distance computation, the bigger $d' = |d(\boldsymbol{q},\boldsymbol{r}) - d(\boldsymbol{p},\boldsymbol{r})|$, the higher the probability of d' exceeding similarity threshold. Thus a proper selection criterion for ***PP*** is that $d' = |d(\boldsymbol{q},\boldsymbol{PP}) - d(\boldsymbol{p},\boldsymbol{PP})|$ should be as large as possible. Let $\boldsymbol{X} \in \mathcal{R}^d$ be the dataset in a bucket and query $\boldsymbol{q}$ is extracted from the distribution of ***X***, the quality of a pivot point ***PP*** can be measured by the Mean Difference between Distances of each data point $\boldsymbol{p}$ and query $\boldsymbol{q}$ to ***PP*** (MDD):

$$\mathrm{MDD}(\boldsymbol{PP}) = \mathrm{E}(|d(\boldsymbol{q},\boldsymbol{PP}) - d(\boldsymbol{p},\boldsymbol{PP})|) \qquad (3)$$

$$= \iint_{\boldsymbol{q},\boldsymbol{p}\in\boldsymbol{X}} p(\boldsymbol{q},\boldsymbol{p})|d(\boldsymbol{q},\boldsymbol{PP}) - d(\boldsymbol{p},\boldsymbol{PP})|\mathrm{d}\boldsymbol{q}\mathrm{d}\boldsymbol{p} \qquad (4)$$

where $\boldsymbol{q},\boldsymbol{p} \in \boldsymbol{X}$ and $p(\boldsymbol{q},\boldsymbol{p})$ is the probability distribution of $(\boldsymbol{q},\boldsymbol{p})$. Fig. 2 illustrates how to choose ***PP*** (red points): the mean difference between distances of each data point to ***PP*** that chosen in the line with direction *dir1*, is larger than that chosen in the line with direction *dir2*. We call the direction of the line through ***PP*** and mean vector of ***X*** as Main Direction (denoted by **MD** in the following), with direction vector $\boldsymbol{\omega}$, and we get an optimization problem:

$$\boldsymbol{PP} = \arg\max_{\boldsymbol{PP}\in\mathcal{R}^d} \mathrm{E}(|d(\boldsymbol{x_1},\boldsymbol{PP}) - d(\boldsymbol{x_2},\boldsymbol{PP})|) \qquad (5)$$

where $\boldsymbol{x_1},\boldsymbol{x_2} \in \boldsymbol{X}$. We could shift ***PP*** to the origin and all data points by subtracting ***Vpp*** without changing the value of (5). Let $\boldsymbol{X}^* = \{\boldsymbol{x}^* = \boldsymbol{x} - \boldsymbol{Vpp}: \boldsymbol{x} \in \boldsymbol{X}\}$ be the shifted dataset and (5) can be written as:

$$\boldsymbol{PP} = \arg\max_{\boldsymbol{PP}\in\mathcal{R}^d} \mathrm{E}(|\|\boldsymbol{x_1^*}\|_2 - \|\boldsymbol{x_2^*}\|_2|) \qquad (6)$$

where $\boldsymbol{x_1^*},\boldsymbol{x_2^*} \in \boldsymbol{X}^*$. Since these data points are in the same bucket, the distribution scale in some directions is relatively small (as *dir2* in Fig. 2). If ***PP*** is far from these data points, as shown in Fig. 3, we would have $\theta_1,\theta_2 \to 0$ and make an approximation of $\|\boldsymbol{x_1^*}\|_2 - \|\boldsymbol{x_2^*}\|_2$:

$$|\|\boldsymbol{x_1^*}\|_2 - \|\boldsymbol{x_2^*}\|_2| = \left|\frac{P_1\cos\theta_2 - P_2\cos\theta_1}{\cos\theta_1 * \cos\theta_2}\right| \qquad (7)$$

$$\approx \left|\frac{P_1 - P_2}{\cos\theta_1 * \cos\theta_2} - o(P_1 - P_2)\right| \qquad (8)$$

$$\xrightarrow{\theta_1,\theta_2\to 0} |P_1 - P_2| \qquad (9)$$

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

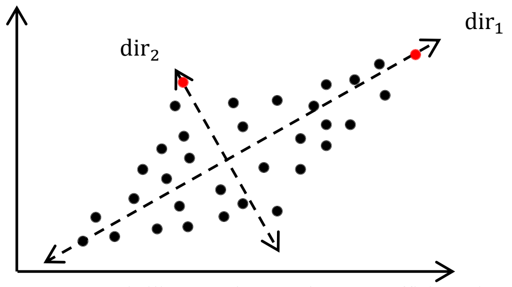


Fig. 2. An example illustrates how to choose an efficient Pivot Point.

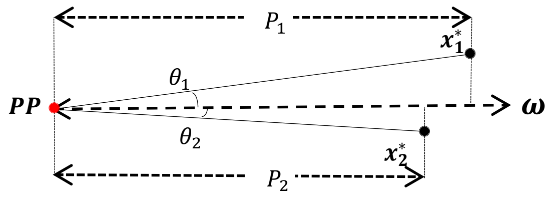


Fig. 3. Approximation of difference of distances when Pivot Point is far away.

Equation (9) means $\|\boldsymbol{x}_1^*\|_2 - \|\boldsymbol{x}_2^*\|_2$ is proportional to the difference between their projections to $\boldsymbol{\omega}$ when ***PP*** is far away. Thus, to maximize MDD(***PP***) is to maximize $\mathrm{E}(|\boldsymbol{x}_1^* \cdot \boldsymbol{\omega} - \boldsymbol{x}_2^* \cdot \boldsymbol{\omega}|)$ when ***PP*** is far away. After the shift, $\boldsymbol{\omega}$ is the only factor and (6) can be written as:

$$\boldsymbol{\omega} = \arg\max_{\boldsymbol{\omega}\in\mathcal{R}^d} \mathrm{E}(|\boldsymbol{x}_1^* \cdot \boldsymbol{\omega} - \boldsymbol{x}_2^* \cdot \boldsymbol{\omega}|) \qquad (10)$$

where $\boldsymbol{x}_1^*, \boldsymbol{x}_2^* \in \boldsymbol{X}^*$. 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 $\boldsymbol{\omega}$. From Cauchy–Schwarz inequality we have:

$$\mathrm{E}(|\boldsymbol{x}_1^* \cdot \boldsymbol{\omega} - \boldsymbol{x}_2^* \cdot \boldsymbol{\omega}|) \le \left(\mathrm{E}(|\boldsymbol{x}_1^* \cdot \boldsymbol{\omega} - \boldsymbol{x}_2^* \cdot \boldsymbol{\omega}|^2)\right)^{\frac{1}{2}} \quad (11)$$

Equation (11) gives an upper bound of MDD. Maximizing the upper bound would guarantee a high probability to maximize the related MDD. It is easy to prove

$$\mathrm{E}(|\boldsymbol{x}_1^* \cdot \boldsymbol{\omega} - \boldsymbol{x}_2^* \cdot \boldsymbol{\omega}|^2) = \mathrm{E}(|y_1^* - y_2^*|^2) = 2\mathrm{Var}(Y^*) \quad (12)$$

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

$$\mathrm{Var}(Y^*) = \mathrm{Var}(Y) = \mathrm{Var}(\boldsymbol{x} \cdot \boldsymbol{\omega}) \qquad (13)$$

where $\boldsymbol{x} \in \boldsymbol{X}$. From Principal Components Analysis [8], 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 will be largest. Thus $\boldsymbol{\omega}$ can be obtained by solving the following equation:

$$\boldsymbol{S}\boldsymbol{\omega} = \lambda\boldsymbol{\omega} \qquad (14)$$

where $\boldsymbol{S}$ is the Covariance Matrix of $\boldsymbol{X}$. The direction vector of eigenvector of $\boldsymbol{S}$ with largest eigenvalue is selected as $\boldsymbol{\omega}$, and ***PP***, as shown in Fig. 4, is:

$$\boldsymbol{PP} = \overline{\boldsymbol{x}} + L * \boldsymbol{\omega} \qquad (15)$$

where $\overline{\boldsymbol{x}}$ is the mean vector of $\boldsymbol{X}$ and L is a scalar. In order for ***PP*** to be far from $\boldsymbol{X}$, L should be large enough. In our following experiment, L is set to $4\|\overline{\boldsymbol{x}}\|_2$.

Larger improvement can be obtained by using multi-pivots. We can use the eigenvector with the second largest eigenvalue to get another pivot. In Section IV, our experiment shows that by using multi-pivots, the query improvement is higher.

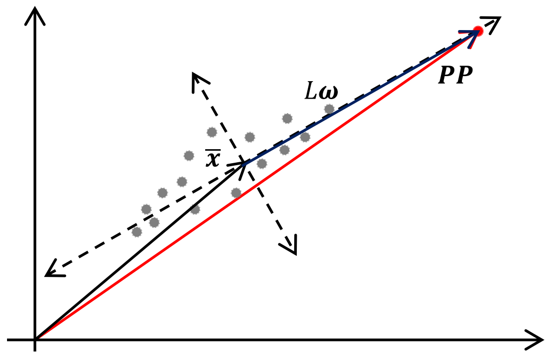


Fig. 4. An example illustrates the relationship between **Pivot Point**, $\overline{\boldsymbol{x}}$ and $\boldsymbol{\omega}$.

C. Pivot-Based Filtering Algorithm

After constructing LSH index, for each bucket, invoke **UpdateIndex** to get ***PP*** and compute the distance between ***PP*** and each data point. Algorithm **UpdateIndex** is illustrated in Algorithm 1.

ALGORITHM 1: **UpdateIndex**

```
Input: a data bucket, the dataset in it is X
1.  Calculate the Covariance Matrix:
        $\boldsymbol{S} = \frac{1}{N}\sum_i \boldsymbol{x}_i \cdot \boldsymbol{x}_i^T - \overline{\boldsymbol{x}} \cdot \overline{\boldsymbol{x}}^T$
    where $N$ is the number of points, $\boldsymbol{x}_i \in \boldsymbol{X}$ and $\overline{\boldsymbol{x}}$ is the
    mean vector of $\boldsymbol{X}$. Choose the eigenvector with the largest
    eigenvalue as $\boldsymbol{\omega}$.
2.  Set
        $\boldsymbol{PP} = \overline{\boldsymbol{x}} + L * \boldsymbol{\omega}$
3.  Compute the distance between PP and each data point.
Output: an updated data bucket
```

Invoke algorithm **QueryFiltering** to process a query. It is illustrated in Algorithm 2. For Range query, the similarity threshold is similar radius; for (Approximate) Nearest Neighbor query, the threshold is the distance between $\boldsymbol{q}$ and the nearest neighbor encountered so far.

ALGORITHM 2: **QueryFiltering**

```
Input: a query q
1.  Hash $\boldsymbol{q}$ to the corresponding bucket in each hash table.
2.  For each data point $\boldsymbol{p}$ in the bucket, first calculate:
        $d' = |d(\boldsymbol{q},\boldsymbol{PP}) - d(\boldsymbol{p},\boldsymbol{PP})|$
3.  If $d'$ exceeds the similarity threshold, skip $\boldsymbol{p}$ and go to 2
    for the next data point.
4.  Else calculate $d(\boldsymbol{q},\boldsymbol{p})$ to determine whether $\boldsymbol{p}$ is similar
    with $\boldsymbol{q}$. Go to 2 for the next data point.
Output: Query result
```

In the following section, we conduct a series of experiments to test our algorithm, and these experiments illustrate the effectiveness of our algorithm.

IV. EXPERIMENT RESULT

In practical applications, the most common similarity search problems are Range query and (Approximate) Nearest Neighbor query. In this section, we conduct a series of experiments to test our algorithm. Our objective is to verify the speedup of filtering process of Euclidean LSH obtained by our algorithm. The experiments are done on a PC with a 32-bit 2.6GHz CPU and 4GB RAM.

A. Range Query

The benchmark dataset we used is the Audio Data in [9]. The search set contains 54,387 192-dimensional data points, and we randomly choose 1,000 points from query set as our query set. Since each hash table in LSH is independent, the effectiveness of speedup can be observed just by one hash table, thus we use one hash table and set the dimension of hash value K=6. For each query point $\boldsymbol{q}$, we record the number of points skipped (the actual distance of these points to $\boldsymbol{q}$ is not calculated) when searching data point $\boldsymbol{p}$ in the dataset that $d(\boldsymbol{p},\boldsymbol{q}) \le r$, where r is the similar radius, and calculate the speedup of filtering process. We increase r gradually and set W=4r to get best performance. With different Pivot Point selection, the speedup is averaged among 1000 queries. As shown in Fig. 5, by using our algorithm, the filtering process is significantly accelerated: the speedup of filtering process obtained by data-based pivot is about 200%, higher than randomly selected pivot (about 140%). While using multi-pivots (2 pivots), the speedup is even higher, almost 300%.

B. *Approximate Nearest Neighbor Query*

The benchmark dataset [10] used to illustrate the effectiveness for ANN query is the INRIA Holidays dataset, consisting of 128-dimensional SIFT descriptors. We randomly choose 100,000 data points from the search set as our search set and 1,000 data points from the query set as our query set, and all these data points are normalized by dividing each dimension by the largest norm in search set. We set the dimension of hash value K=5 and use one hash table. For each query point ***q***, the number of points skipped when searching the nearest neighbor ***p*** in search set is recorded. We only use one data-based pivot and the speedup is averaged among 1000 queries under different W. As shown in Fig. 6, the speedup of ANN query is almost 500%, which means the exact distance computation is significantly reduced.

Another important similarity search problem is k-Approximate Nearest Neighbor query (k-ANN), which is an important variant of ANN query. Our method is able to speed up ANN query, leading to the potential to speed up k-ANN query.

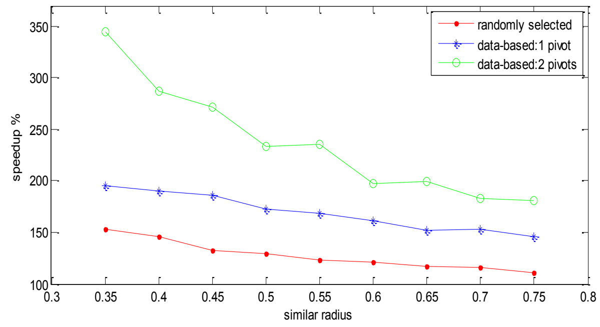


Fig. 5. Speedup of filtering process obtained by our algorithm for Range query with different pivot selection. The horizontal ordinate is similar radius and the vertical ordinate is percentage of speedup. The data-based pivot is effective than randomly selected pivot, while multi-pivots is more effective.

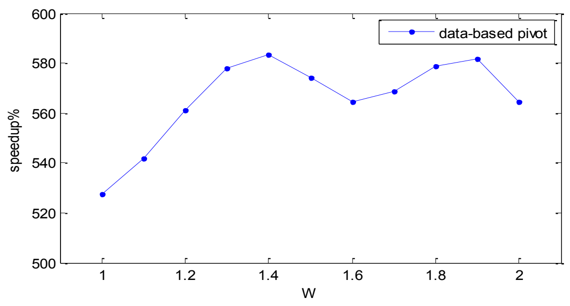


Fig. 6. Speedup of filtering process obtained for Approximate Nearest Neighbor query by using one data-based pivot. The horizontal ordinate is W, the vertical ordinate is percentage of speedup. The speedup is almost 500% which means the exact distance computation reduced is significant.

From Fig. 5 and Fig. 6, we can conclude that by using our algorithm, the query process of LSH is almost 5 times faster for ANN query and 2-3 times faster for Range query. When the dataset is large-scale and high-dimensional, the exact similarity computation reduced is significant while the memory enlargement compared with the original memory occupation is negligible, e.g. when using E2LSH with 8-d hash value and 20 hash tables to query a 128-d SIFT dataset containing 1,000,000 data points, the original memory occupation is at least $4\times(128+8\times20)\times1{,}000{,}000$ bytes while the enlargement is $4\times20\times1{,}000{,}000$ bytes, only 6.9% of the original memory occupation.

V. CONCLUSION

In this paper, we analyze the phenomenon "Non-Uniform" that degrades the query performance of Euclidean LSH. "Non-Uniform" will cause exhaustive computation in the filtering process of LSH and significantly degrade the query performance, especially when the dataset is large-scale. We propose a pivot-based algorithm to accelerate the filtering process and also provide a method to get optimal pivot. Our algorithm is simple, and experiments show that by using our algorithm, with little memory enlargement, the filtering process is almost 5 times faster for ANN query and 2-3 times faster for Range query, which means the cost of computation is significantly reduced. Moreover, since we make no assumption about the dataset, our method can be applied to most Euclidean LSH-based index structures [4], [5].

REFERENCES

[1] P. Indyk and R. Motwani. "Approximate nearest neighbor: towards removing the curse of dimensionality," Proceedings of the Symposium on Theory of Computing, pages 604-613, 1998.

[2] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," CACM, 51, 1 (2008), pages 117-122.

[3] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," In Proc. of the 20th Symposium on Computational Geometry(SCG), 2004.

[4] R. Panigrahy, "Entropy based nearest neighbor search in high dimensions," In Proc. of ACM-SIAM Symposium on Discrete Algorithms(SODA), pages 1186–1195, 2006.

[5] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe lsh: efficient indexing for high-dimensional similarity search," In Proc. of VLDB, pages 253-262, 2007.

[6] A. Joly and O. Buisson, "A posteriori multi-probe locality sensitive hashing," In Proc of ACM MM, 2008.

[7] E. Cháves, G. Navarro, R. Baeza-Yates, and J.L. Marroquín,"Searching in metric spaces," ACM Computing Surveys, 33(3):273–321, 2001.

[8] I.T. Jolliffe. Principal Component Analysis, Series: Springer Series in Statistics, 2nd ed., Springer, NY, 2002, XXIX, 487 p. 28 illus. ISBN 978-0-387-95442-4.

[9] W. Dong, Z. Wang, W. Josephson, M. Charikar, and K. Li, "Modeling lsh for performance tuning," In CIKM, 2008.

[10] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Y. Wu, "An optimal algorithm for approximate nearest neighbor searching," J. of the ACM, 45:891–923, 1998.