Data-Oriented Locality Sensitive Hashing

Wei Zhang[1,2], Ke Gao[1], Yong-dong Zhang[1], and Jin-tao Li[1]
[1]Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[2]Graduate University of the Chinese Academy of Sciences, Beijing, China
{zhangwei, kegao, zhyd, jtli}@ict.ac.cn

ABSTRACT
Locality Sensitive Hashing (LSH) has been proposed as a scalable and high-dimensional index for approximate similarity search. Euclidean LSH is a variation of LSH and has been successfully used in many multimedia applications. However, hash functions of the basic Euclidean LSH project data points over randomly selected directions, which reduces accuracy when data are non-uniformly distributed. So more hash tables are needed to guarantee the accuracy, and thus more memory is consumed. Since heavy memory cost is a significant drawback of Euclidean LSH, we propose Data-Oriented LSH to reduce memory consumption when data are non-uniformly distributed. Most of existing methods are query-directed, such as multi-probe and query expansion methods. We focused on the hash table construction, and thus the query-directed methods can be applied to our index to improve further. The experiment shows that to achieve the same accuracy, our method uses less time and less memory compared with original Euclidean LSH.

Categories and Subject Descriptors
H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval

General Terms
Algorithm, Performance

Keywords
Similarity Search, Approximate Nearest Neighbor Search, Data-Oriented Locality Sensitive Hashing

1. INTRODUCTION
High dimensional data indexing and similarity search are essential for building scalable content-based search systems on feature-rich multimedia data. Index structure is challenged to be efficient and effective to organize large scale high-dimensional data. Approximate nearest neighbor methods have been proposed to achieve low on-line query costs. Among these methods, Locality Sensitive Hashing (LSH) [1, 2] is well known for its sub-linear query time and controllable success probability.

MM'10, October 25–29, 2010, Firenze, Italy.

LSH is efficient to organize and query large-scale and high-dimensional database and has been successfully used in local features indexing and 3D object indexing [3]. The best known variation of LSH is Euclidian LSH. To reduce the memory consumption of Euclidian LSH [2], many improved versions [3, 4] have been proposed, which reduce hash tables used for indexing and guarantee accuracy by multi-probing buckets in one hash table. All these methods are query directed and based on the basic Euclidian LSH index structure.

The principle of LSH is that nearby data points are hashed into the same bucket with a high probability while points faraway are hashed into the same bucket with a low probability. For the basic Euclidean LSH, hash functions project data points linearly over randomly selected directions [3]. Then projected values are divided uniformly to generate hash values. For uniformly distributed data, data points are hashed to buckets with equal probability, and thus the points in buckets are uniformly indexed. However, in multimedia applications, the distribution of descriptors is far from being uniform [5]. Consequently, the distribution of randomly projected values is far from being uniform. For example, [3] estimates projected results over randomly directions as Gaussian distributed. Therefore hash values generated by dividing projected values uniformly will be aggregated. In such a situation, querying will cost a lot of time for many disturbance points are probed. To reduce disturbances, the basic LSH concatenates several hash functions together, so the collision probability of far away objects becomes very small, but it also reduces the collision probability of nearby objects. Although the query time is reduced, more hash tables are needed to guarantee accuracy. This heavy memory consumption is the significant drawback of basic Euclidean LSH.

The basic Euclidean LSH selects projecting directions of hash functions blindly, without considering data distribution. Based on the above analysis, we improve hash structure by taking advantage of data distribution information. Firstly, we analyzed influences of data distribution on hash functions of Euclidean LSH and discussed what a good hash function is. Then we proposed measurements to intuitively evaluate different hash functions. Secondly and more significantly, we proposed an improved Data-Oriented LSH index structure by selecting adaptive projecting vectors to generate hash tables, which not only reduces query time but also improves the accuracy of k-nearest neighbors retrieval compared with original Euclidean LSH. Moreover, to achieve the same accuracy as original Euclidean LSH, less hash tables are needed and thus memory consumption is reduced.

Our work focuses on the construction of hash tables, multi-probe and query expansion methods are complementary and can be applied to our index structure to improve further.

2. RELATED WORK

The basic idea of LSH is to map similar data points to the same bucket in a hash table with probability higher than non-similar points. To implement LSH [2], hash function family H is introduced. Formally, A family H= {h: S->U} is called (r_1, r_2, P_1, P_2)-sensitive for D, if for any p, q ∈ S：

$$\text{If } D(p,q) \le r_1 \text{ then } \Pr[h(p)=h(q)] \ge P_1$$
$$\text{If } D(p,q) \ge r_2 \text{ then } \Pr[h(p)=h(q)] \le P_2$$

where probability $P_1 > P_2$, radius $r_1 < r_2$ and D is a similarity measure. Then, function family G is defined as:

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

where $g(p)=(h_1(p),...,h_K(p))$, $K>0$ and $h_i \in H$. A hash table is constructed according to one g of concatenated h. And LSH uses L ($L>0$) hash tables to guarantee the accuracy.

LSH proposed in [2] is in L_p norm and projects high dimensional data to p-stable random vectors as hash functions. A p-stable hash function h is defined as

$$h_{a,b}(x)=\left\lfloor \frac{a \cdot x + b}{w} \right\rfloor \quad (1)$$

Where a is a d-dimensional vector with entries chosen independently from a p-stable distribution and b is a real number chosen randomly from the range $[0, w]$. And h maps vector x to an integer. The most common used p is 2, which is known as Euclidean LSH [2]. Euclidean LSH is efficient, but requires many hash tables and consumes memory heavily. The multi-probe methods [3,4] are query directed and based on the basic Euclidean LSH method using randomly selected projection vectors.

Since $a \cdot x + b$ is divided uniformly by w, the projected values $a \cdot x$ should be as uniform as possible to generate uniform hash values and final index. But the Euclidean LSH projects points over randomly selected directions and projected results over randomly selected directions [3] are estimated as Gaussian distributed. For our dataset (see Section 4), the bottom row of Figure 2 shows that projected values of original hash functions are Gaussian distributed. This will lead to aggregated buckets, for points of the same hash value will be hashed to the same bucket. Thus in order to achieve fast on line query, basic LSH concatenates K hash functions to reduce points in one bucket. However, this also reduces the collision probability of near neighbor points. In order to guarantee the accuracy, L hash tables are used to probe near neighbors in L buckets, which results in the heavy memory consumption.

Based on the above analysis, we conclude that projecting directions are significant to the Euclidean LSH index. In the next Section, we will discuss what a good projecting direction is and propose measurements of good hash functions. Under these measurements, we will show an improved data-oriented hash index.

3. DATA-ORIENTED LSH

3.1 Analysis of Good Projecting Directions

Most of existing methods estimate accuracy and efficiency [3, 4] to assess the constructed index. The quality of index is assessed indirectly, for such estimations are query dataset dependent. To observe the LSH index structure more intuitively and objectively, we propose measurements to evaluate the hash functions that construct the index.

Figure 1 shows sample examples of two projected results of the same 100 data points using different projection vectors in 2-d. The distribution of projected values in the Left is more aggregated and the Right is much more uniform. Based on the analysis in Section 2, a hash function using good projecting directions should satisfy the following two aspects:

Firstly, relative distance is retainable. A good hashing function retains the relative distance of points in original data space as much as possible, which means hashing near points to the same bucket and far points to different buckets with high probability.

Secondly, distribution of hashed points is uniform. A good hashing function distributes hashed points to discriminative buckets proportionally, i.e. hashed points are not aggregated together. This will results in retrieving points in a bucket with less disturbances hence less query time.

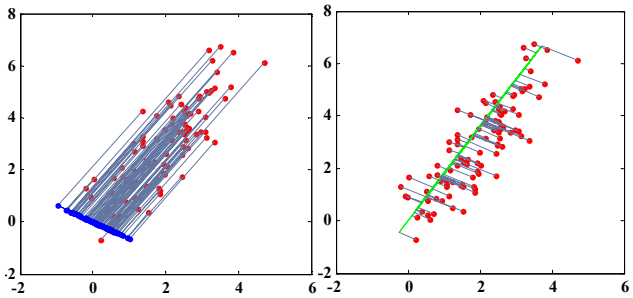

Figure 1. Sample examples of different projections in 2-d.

For the first aspect, precision and recall of k Nearest Neighbors (kNNs) is often used to evaluate query performance. However, in range similarity query the precision is always 1.0 and in kNNs retrieval the precision and recall are identical. And precision and recall measurements treat the missing of the first NN and the k-th NN (k>0) as equal, which is unreasonable. In addition, relative error on distances measured in different datasets cannot be compared. Therefore, we use EP [6], *error on the position*, to assess the query performance. Given an order list S, an object o in S is denoted as S(o). Then EP is defined as:

$$EP = \frac{\sum_{i=1}^{|S|} (OX(O_i) - S(O_i))}{|S| \cdot n} \quad (2)$$

Where S is a result-set returned by kNNs retrieval and ordered according to distances to the query point, OX is the ordered list containing exact kNNs, and n is the size of database.

For the second aspect we propose a new measurement to evaluate the hashing function. For a hash function h that construct a hash table, assume that the quantity of points in bucket i is N(i) and P(N(i)), the probability of points in i, is estimated by N(i)/n. The distribution entropy of h is defined as follows:

$$E(h) = -\sum P(N(i)) * \log(P(N(i))) \quad (3)$$

Higher E(h) indicates better distribution of hashed points, i.e. the higher of E(h), the less disturbances are probed in one bucket.

Since K ($K>1$) hash functions are always used in basic LSH to build one hash table, definition (3) can be extended to analysis

hash functions of g that concatenates h. That is, extend E(h) to the distribution entropy E(g).

3.2 Data-Oriented Hash Functions

In this section, hash functions using data-oriented projecting vectors are introduced. Under the measurements proposed in Section 3.1, the data-oriented hash functions show significant improvement.

PCA (Principal Components Analysis) is a statistical analysis method which reveals the hidden and simplified structure and reduces a complex dataset to a lower dimension. By using PCA to align the axes, we can assume a uniform distribution on each axis. Assume a dataset X of feature descriptors is a vector of $X_1,\ldots, X_n$. And X_i ($1<i<n$) is a d-dimensional descriptor vector. Assume that X has been subtracted mean in each dimension, and the covariance matrix is: $\sum = \frac{1}{n-1} X^{\mathrm{T}} X$. Eigenvectors V is computed in the decreasing order of eigenvalues of $\sum$. The new dataset Y that can represent X is calculated as $Y = XV$. For Y are projections on a new set of basis vectors V, we are motivated to choose V as hash projecting vectors. Then hash functions are generated as (1). V contains data distribution information and is data-oriented. The hash tables using V as projecting vectors are also data-oriented.

Using dataset in Section 4, data-oriented and random hash functions are compared. Figure 2 shows the distribution histograms of hashed values using PCA vectors projection and random vectors projection. The lengths of projecting vectors are normalized to 1.0.

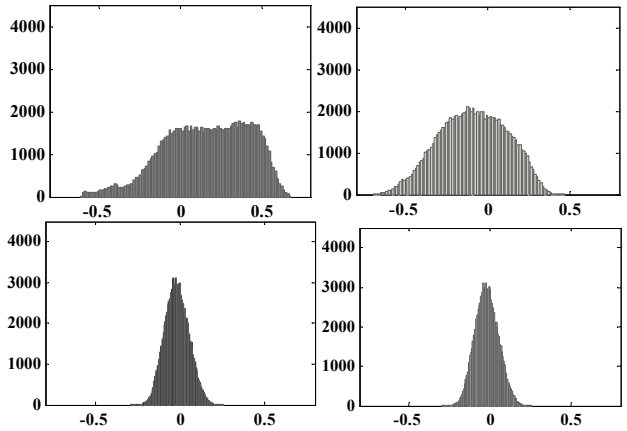

Figure 2. Distribution histograms of projected values over different directions. Horizontal ordinate is the projected values and vertical is the number of points. Top row shows distributions of projected values over the first and second PCA vector respectively. Bottom row shows distributions of random projections

The distributions of projections generated by PCA vectors are much more uniform than random vectors. And more important, PCA vectors retain relative distances in original space as much as possible, hence fewer disturbances are probed. PCA vectors are consistent in both aspects proposed in Section 3.1. More precisely, the distribution entropies of different hash functions computed by formulate (2) are summarized in Table 1. The projection vectors of data-oriented hash functions are selected from the first four components of V. And random hash functions of the basic LSH are generated by (1).

Table 1. Comparison of distribution entropies between Data-Oriented and Random hash functions.

Hash Function	Data-Oriented	Random
hash function 1	5.9593	3.8920
hash function 2	5.7134	3.8921
hash function 3	5.3034	3.8921
hash function 4	5.4140	3.8923

Table 1. shows that the distribution entropies of data-oriented hash functions are much higher than random. The distribution of bucket points in a hash table is improved. We insist that as PCA has been applied broadly in computer vision, the data-oriented method can be applied to many other datasets directly.

3.3 Data-Oriented LSH Index Algorithm

Based on data-oriented hash functions, construction algorithm of the Data-Oriented LSH index is illustrated in Algorithm 1.

Algorithm 1: Construction of Data-Oriented LSH index

Input: Dataset *X*

1. Choose a feature dataset X' generated from a randomly chosen image set. Our experiment shows that when the size of X' is larger than 5000, the computed V (see Section 3.2) makes little difference to generate hash tables.

2. Compute V of X' offline.

3. Hash functions are generated as follows:

Compute the distribution entropy *entropy_random* of hash functions using Gaussian random projected vectors. Choose vectors from V whose distribution entropies are larger than *entropy_random*. Then we get t PCA vectors $V'= [V_1,\ldots V_t]$ in decreasing order of distribution entropy.

4. The L ($L \le t$) hash tables are built as follows:

4.1 Choose a projecting vector V_i ($1 \le i \le t$) from V' in order.

4.2 Compute hash value Hash(x) for each $x \in X$:

(1) Project(x) = xV_i ; (2)Hash(x)= Floor(Project(x)/w). In PCA, data are subtracted mean before projected to new basis. For projecting x subtracted mean over V_i equals moving x in V_i dimension (i.e. (x-*mean*) $V_i = xV_i - meanV_i$) and $meanV_i$ is equal to all x, hence we omit the step of subtracting mean.

4.3 Insert x to bucket according to Hash(x).

Output: Data-Oriented LSH Index

In Algorithm 1, L and w should be optimized according to accuracy and efficiency requirements as the basic LSH method. The similarity query algorithm of data-oriented LSH consists of two steps: (1) finding candidate points that are hashed to the same bucket with the query point in each hash table and (2) ranking the candidate points according to their L_2 distances to the query point.

4. EXPERIMENTAL RESULTS

We demonstrate our method on object recognization benchmark database of University of Kentucky [7], which contains 10,200

images that consists of groups of 4 images of each object. MSER (Maximally Stable Extreme Regions) performs well on a wide range of test sequences [8], so we use MSER to detect local interesting regions. Then we use SIFT [8] to get 128-dimentional local feature descriptors of MSER patches.

The experiments are done on a PC with one 32-bit 2GHz CPU and 1GB RAM. The source code of Euclidean LSH provided by the authors is called E2LSH. Our objective is to verify the improvement of Data-Oriented LSH index over E2LSH. We use a 100K subset of the full SIFT descriptor dataset to evaluate query performance, for E2LSH consumes memory heavily. All results are averaged among 100 queries. In LSH larger parameter w means that more points are hashed to one bucket, which also means lower precision, higher recall and more query time. We will increase w gradually and choose the best K for w in the following experiments.

To demonstrate the differences between recall and EP, we retrieved 10 NNs from one hash table using data-oriented LSH. For convenience, 1.0-Recall is compared with EP in Figure 3. The red rectangle contains the area that 1.0-Recall increases slightly and EP decreases rapidly, which indicates that though we lost some exact 10 NNs, we get more valuable nearest neighbors that are much nearer to query points. The experiment shows that EP evaluates hash structure more reasonably than recall.

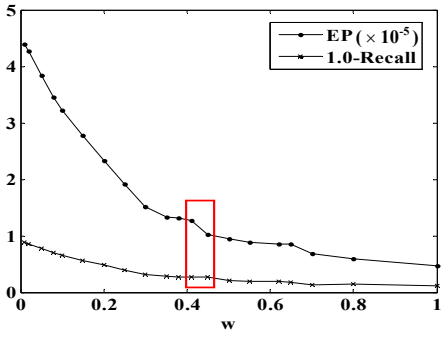

Figure 3. Comparison between 1.0-Recall and EP of 10 NNs.

To show the improvement of data-oriented hash functions, we compared our method with E2LSH using EP and recall of 10 NNs. To reduce the exact L_2 distance computation, we introduced the triangle inequality to filter more points. With carefully chosen searching radius, the filter works well and reduces much time. In this experiment, we set a large radius to get all 10 NNs, which increases some query time.

From Figure 4, we conclude that using equal amounts of hash tables, with the same EP, Data-Oriented LSH uses less time and the time is reduced by about 33% on average. While with the same querying time, Data-Oriented LSH has smaller EP and the EP is reduced by about 15% on average. The EP-time comparison result is basically consistent with recall-time. With the same recall, Data-Oriented LSH uses less hash tables than E2LSH. We observed that the reduced EP by increasing w is comparable to and even better than increasing L. The reason is that more relative distances of points are retained by data-oriented projections than randomly projections. And this indicates that multi-probe method can be used to gain further improvement.

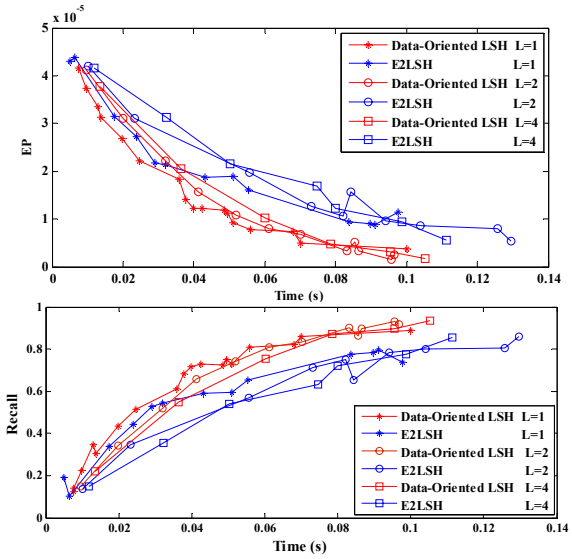

Figure 4. Comparison of query performance between Data-Oriented LSH and E2LSH.

5. CONCLUSION

In this paper, we presented a new Data-Oriented LSH index which reduces heavy memory cost of Euclidean LSH caused by randomly selected projecting directions. And we also proposed EP and distribution entropy to analyze and evaluate hash index. Experiments show that with the same accuracy, Data-Oriented LSH advances E2LSH in both time and memory consumption.

6. ACKNOWLEDGMENTS

This work was supported by the National Nature Science Foundation of China (60802028, 60873165), National Basic Research Program of China (973Program, 2007CB311100), Co-building Program of Beijing Municipal Education Commission.

7. REFERENCES

[1] Andoni, A. and Indyk, P. Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. CACM, 51, 1 (2008), 117-122.

[2] Datar, M., Immorlica, N., Indyk, P. and Mirrokni, V. Locality-sensitive hashing scheme based on p-stable distributions. SCG '2004.ACM Press.

[3] Lv, Q., Josephson, W., Wang, Z., Charikar, M. and Li, K. Multi-probe LSH: efficient indexing for high-dimensional similarity search. VLDB, 2007.

[4] Joly, A. and Buisson, O. A posteriori multi-probe locality sensitive hashing. In MM, 2008.

[5] Poullot, S., Buisson, O. and Crucianu, M. Z-grid-based probabilistic retrieval for scaling up content-based copy detection. CIVR, 2007.

[6] Zezula, P., Amato, G., Dohnal, V. and Batko, M. Similarity Search: The metric space approach. Advances in Database Systems, 32, (2006).

[7] D. Nistér and H. Stewénius. Scalable recognition with a vocabulary tree. CVPR. 2(2006), 2161-2168.

[8] Mikolajczyk, K., Tuytelaars, T., Schmid, et al. A comparison of affine region detectors. IJCV, 65, 1(2005), 43-72.