2012 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing

Query Range Sensitive Probability Guided Multi-Probe Locality Sensitive Hashing

Xiaoguang Gu[1,2,3], Lei Zhang[1,2,3], Dongming Zhang[1,3], Yongdong Zhang[1,3], Jintao Li[1,3], Ning Bao[4]
[1]Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[2]Graduate University of Chinese Academy of Sciences, Beijing, China
[3]Beijing Key Laboratory of Mobile Computing and Pervasive Device, Beijing, China
(Institute of Computing Technology, Chinese Academy of Sciences)
[4]School of Technical Physics, Xidian University, Xi'An, China
xggu@ict.ac.cn

***Abstract*—Locality Sensitive Hashing (LSH) is proposed to construct indexes for high-dimensional approximate similarity search. Multi-Probe LSH (MPLSH) is a variation of LSH which can reduce the number of hash tables. Based on the idea of MPLSH, this paper proposes a novel probability model and a query-adaptive algorithm to generate the optimal multi-probe sequence for range queries. Our probability model takes the query range into account to generate the probe sequence which is optimal for range queries. Furthermore, our algorithm does not use a fixed number of probe steps but a query-adaptive threshold to control the search quality. We do the experiments on an open dataset to evaluate our method. The experimental results show that our method can probe fewer points than MPLSH for getting the same recall. As a result, our method can get an average acceleration of 10% compared to MPLSH.**

Keywords- Approximate Similarity Search; Locality Sensitive Hash; Multi-Probe; Query Range Sensitive

I. INTRODUCTION

Locality Sensitive Hashing [1][3][4][8][16] is proposed to construct the effective and efficient indexes for high-dimensional approximate similarity search. LSH has been successfully used for image retrieval [12][13] and 3D object indexing [14][15]. Although LSH can get perfect results in theory, the limitation of LSH is that its memory consumption is too large. The reason is that many hash tables are needed to improve recall. In [1] and [8], a large number of hash tables are used. If the number of hash tables is large, the index structure cannot be loaded in main memory to get the best performance. Some variations [5][7][9] based on multi-probe strategy have been brought forward to reduce the number of hash tables .

The first multi-probe strategy is proposed by Entropy Based LSH [5]. Through randomly generating the 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.

Multi-Probe LSH [7] is inspired by and improves upon Entropy Based LSH and query adaptive method [10]. A more efficient algorithm is proposed to generate optimal probe sequence of hash buckets which are likely to contain similar points to the query.

Unlike Multi-Probe LSH which is based on a likelihood criteria, Posteriori Multi-Probe LSH [9] put forward a more reliable posteriori model taking account some prior 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 algorithms proposed in [5][7][9] are excellent for searching approximate top-k nearest neighbors. However, for range queries, the goal of multi-probe should be to find all points in the query range. In this situation, the query range is an important prior knowledge for multi-probing. Unfortunately, the existing multi-probe algorithms do not use this prior knowledge. In this paper, a new probability model taking account the query range and a query-adaptive algorithm are proposed to generate the optimal multi-probe sequence for range queries. We name our method as Query Range Sensitive Probability Guided Multi-Probe LSH (QRSP-MPLSH). The contributions of this paper are shown as follows.

1. Proposing a novel probability model to guide multi-probe. The probability model proposed in this paper uses the query range as a parameter to compute the probability, so the probability can reflect the actual possibility of a bucket containing the points in the query range.

2. Putting forward a query-adaptive threshold to control the multi-probe procedure to avoid probing too few or too many candidate buckets.

II. BACKGROUND

A. Locality Sensitive Hash (LSH)

The principle of LSH [1] is to project similar data points to the same bucket with a higher probability than dissimilar points. In [3], a more practicable LSH scheme based on p-stable distributions is presented for l_p norm. We remind this scheme here, since our work is based on the same scheme. For the l_2 metric, the typical used LSH function is defined as:

$$h(v)=\left\lfloor \frac{a\cdot v+b}{w} \right\rfloor \qquad (1)$$

where a is a d-dimensional random vector with entries chosen independently from a Gaussian distribution and b is a real number chosen uniformly from the range $[0, w]$. w is a positive real number. E2LSH [4] is the more practical approach for l_2 norm.

978-0-7695-4761-9/12 $26.00 © 2012 IEEE
DOI 10.1109/SNPD.2012.35

B. Range Query Using Basic LSH

Given a query point q and a range R, the goal of range query is to find all points within the distance R from q. When L hash tables are constructed, range query can be done as follows:

1. Compute the L hash values of a given query point to generate L candidate hash buckets, one in each hash table.
2. Collect all candidate points in L candidate hash buckets.
3. Filter all candidate points by computing their distance to the query point to find the target points which are in the query range.

For multi-probe method, in second step, more than L candidate hash buckets will be probed.

C. Multi-Probe

For a query point q, its hash bucket is $g(q)=(h_1(q),\ldots,h_k(q))$ [1]. Define the hash perturbation vector to be $\Delta=(\delta_1,\ldots,\delta_k)$. According to [7], we can believe that the bucket whose vector is little different from the query point's vector may contain the neighbor points with a high probability. If w is reasonably large, the neighbor points should be hashed into the same or adjacent interval by $h(v)$. Hence, we can restrict $\delta_j \in \{-1,0,1\}$ (j =1…k). The set of all candidate buckets to be probed is defined as follows.

$$PB=\{v_q : v_q = g(q)+\Delta\} \tag{2}$$

The total number of the candidate buckets is 3^k. In practice, we cannot probe all these buckets. Only the buckets with a high probability of containing the neighbor points should be probed.

III. Our Multi-Probe Method

A. Query Range Sensitive Probability Model

In the following, we propose a probability model to predict how possible a candidate bucket holds the points within the query range R.

For two arbitrary vectors p, q in n-dimensional feature space, we can compute their hash bucket $g(p)=(h_1(p),h_2(p),\ldots h_k(p))$ and $g(q)=(h_1(q),h_2(q),\ldots,h_k(q))$. h_j is defined as formula (1). By h_j, n-dimensional feature point is projected on a_j. Then the projected value is quantized by w. We define the un-quantized hash value as:

$$f_j(q)=a_j\cdot q+b_j \tag{3}$$

Like [7], using $x_j(-1)$ and $x_j(1)$ to represent the distances from $f_j(q)$ to the left and the right boundaries of the interval, we can compute the values of $x_j(-1)$ and $x_j(1)$ as follows.

$$x_j(-1)=f_j(q)-w*h_j(q) \tag{4}$$

$$x_j(1)=w-x_j(-1) \tag{5}$$

From formula (3), we can get

$$f_j(p)-f_j(q)=a_j\cdot(p-q) \tag{6}$$

a_j is a vector whose every dimension is randomly drawn out from Normal Distribution. Normal Distribution is a 2-stable distribution [3]. So $f_j(p)-f_j(q)$ follows the normal distribution with parameters $u=0, \sigma^2=c\|p-q\|_2^2$, c is a constant. $\|p-q\|_2$ is the l_2 distance between p and q. We abbreviate $\Pr(h_j(p)=h_j(q)-1)$ as $\Pr_j(-1)$, $\Pr(h_j(p)=h_j(q)+1)$ as $\Pr_j(1)$ and $\Pr(h_j(p)=h_j(q))$ as $\Pr_j(0)$. If we define $\|p-q\|_2$ as r, the probability of p, q falling into the same or neighbor intervals can be calculated as follows.

$$\Pr_j(-1)=\int_{-\infty}^{-x_j(-1)} N(0,\ cr^2)dt \tag{7}$$

$$\Pr_j(1)=\int_{x_j(1)}^{+\infty} N(0,\ cr^2)dt \tag{8}$$

$$\Pr_j(0)=1-\Pr_j(-1)-\Pr_j(1) \tag{9}$$

When q and w are given, $x_j(-1)$ and $x_j(1)$ are constants. $\Pr_j(\delta)$ (δ = -1, 0 , 1) becomes the function of $r=\|p-q\|_2$.

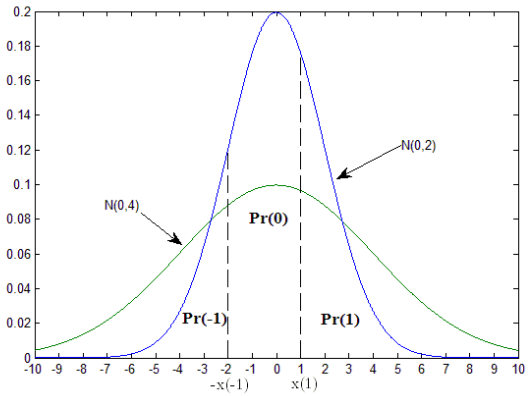

Figure 1. The sensitivity of Prj(δ) to the query range R. When R changes, Prj(-1), Prj(1) and Prj(0) will change accordingly.

r is a random variable and its distribution is unknown. However, our object is to find the points whose distance away from the query point are less than the query range R. Consequently, we can restrict r <=R. When r = R, $\Pr_j(-1)$ and $\Pr_j(1)$ get the maximal values, and $\Pr_j(0)$ get the minimal value. That is to say, the point whose distance away from the query point is R has the maximal probability of falling into the query point's neighbor interval. However, the points within the query range have the less probability of falling into the query point's neighbor interval but the higher probability of falling into the same interval of the query

point. This can be seen in Fig. 1. As a result, to compute $\Pr_j(\delta)$, we can set r=R. In this situation, $\Pr_j(\delta)$ is dependent on R. For the same query point, if R is different, $\Pr_j(\delta)$ is also different. For example, when R increases, $\Pr_j(-1)$ and $\Pr_j(1)$ increase, but $\Pr_j(0)$ decreases. The probability of a candidate bucket containing the points within the query range can be calculated as follows.

$$\Pr(v_q) = \prod_{j=1}^{k} \Pr_j(\delta_j), \delta_j \in \{-1,0,1\} \quad (10)$$

B. Generating Multi-Probe Sequence

To generate the optimal probe sequence, the probabilities of all candidate buckets must be computed and then sorted in decreasing order. It is unpractical to compute all these probabilities for every query point. The algorithm is needed to directly generate the sorted probe sequence without computing every candidate bucket's probability.

For a query point, $\Pr_j(\delta)$ (j *from* 1 *to* $k, \delta \in \{-1,0,1\}$) can be computed according the formulas (7) to (9). For a single hash table, we can get $3k$ values. These $3k$ values are arranged in a 3xk matrix A as (11). In A, every column represents a decreasing order list of $\Pr_j(-1)$, $\Pr_j(0)$ and $\Pr_j(1)$. At the same time, all columns are sorted to decreasing order according to their first elements. That is to say, the first row of A includes all max elements of all columns and all elements in the first row are sorted in decreasing order.

$$A = \begin{bmatrix} a_{01} & a_{02} & \cdots & a_{0k} \\ a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \end{bmatrix}, a_{ij} = \Pr_j(\delta_j(i)) \quad (11)$$

In (11), $\delta_j(i)$ represents the value of δ corresponding to the ith largest probability in $\Pr_j(-1)$, $\Pr_j(0)$ and $\Pr_j(1)$. For example,if $\Pr_j(0) > \Pr_j(-1) > \Pr_j(1)$ then $\delta_j(0)=0$, $\delta_j(1)=-1$ and $\delta_j(2)=1$.

Let z_j represents the order number of the elements in *jth* column of matrix A. Based on matrix A, every perturbation vector Δ can be represented by the key:

$$Z = (z_1, z_2, \cdots z_k), \quad z_j \in \{0,1,2\}, \ 1 \le j \le k \quad (12)$$

The subscript of z_j represents the column-coordinate of matrix A, and the value of z_j represents the row-coordinate of matrix A. Then all elements of A can be described as $a_{z_j j} = \Pr_j(\delta_j(z_j))$. For example, when k = 5, given $Z = (z_1, z_2, z_3, z_4, z_5) = (0,1,2,0,0)$, it corresponds to the vector:

$< a_{01}, a_{12}, a_{23}, a_{04}, a_{05} >$

$=< Pr_1(\delta_1(0)), Pr_2(\delta_2(1)), Pr_3(\delta_3(2)), Pr_4(\delta_4(0)), Pr_1(\delta_5(0)) >$

Consequently, we can get the perturbation vector Δ :

$$\Delta = (\delta_1(0), \delta_2(1), \delta_3(2), \delta_4(0), \delta_5(0))$$

There is a respective value of Z for every candidate bucket defined in formula (2). That is to say, every candidate bucket can be represented by Z, and the probability of every candidate bucket can be computed using Z and matrix A. The formula (10) can be rewritten as:

$$\Pr(v_q) = \Pr(Z) = \prod_{j=1}^{k} a_{z_j j} \quad (13)$$

From the formula (12), we can easily conclude that $Z_0 =< \underbrace{0, 0, \cdots, 0}_{k} >$ corresponds to the maximal probability, and $Z_{3^k-1} =< \underbrace{2, 2, \cdots, 2}_{k} >$ corresponds to the minimal probability. In fact, there must be a unique order of Z that can sort all the probabilities in a deceasing order. The algorithm presented in [9] can be used to get this ordered list.

Until now, we get an ordered list of the candidate buckets. The candidate bucket at the top of the list should be probed firstly. However, not all the 3k buckets must be probed. We define an adaptive threshold to terminate the multi-probe procedure.

$$Threshold = \frac{1}{N} \Pr_{\max} \quad (14)$$

$\Pr_{\max}$ is the maximal probability value of all the candidate buckets for the query point. N is a constant integer. This threshold is adaptive to the query point. Different query point will generate different threshold. As a result, different query point will have different number of probe steps. N can control the query quality. If the value of N increases, the value of threshold decreases. Consequently, more buckets can be probed. Probing more buckets can improve the recall. N can be selected on a training set.

IV. Experimental Setup

A. Experimental Dataset

Define abbreviations and acronyms the first time they are used in the text, even after they have been defined in the abstract. Abbreviations such as IEEE, SI, MKS, CGS, sc, dc, and rms do not have to be defined. Do not use abbreviations in the title or heads unless they are unavoidable.

We use an open dataset to evaluate our method. It is provided by TEXMEX Research Team [11]. The test dataset is summarized in Table 1. A query set containing 10,000 points is provided with the test dataset.

TABLE 1. TEST DATASET SUMMARIZE

Name	#Vectors	#Dimension	Size
ANN_SIFT1M	1,000,000	128	161MB

B. Evaluation Metrics

Precision and recall are often used to evaluate the query quality of all kinds of index methods. However, for range query, the precision is always 1. Hence, we cannot use precision as a metric. In our experiment, we use recall as an

effective metric. Given a query point q, let *B(q)* be the set of all points in the query range *R*, let *A(q)* be the set of points returned by LSH. We define recall as follows:

$$recall = \frac{|A(q) \cap B(q)|}{|B(q)|} \quad (15)$$

However, only using recall is not enough to evaluate multi-probe method. Two different multi-probe methods can get the same recall with the same number of hash tables. Recall cannot effectively distinguish their performance in such case. Although the recall is same, the points probed by different methods may be quite different because different multi-probe methods generate different probe sequences. In this case, we think that one multi-probe based method is better than another if it can probe fewer invalid points but get more valid points. Based on this idea, we bring forward an evaluation metric, named as Valid Probe Ratio, abbreviated as VPR. We define VPR as follows:

$$VPR = \frac{|H(q)|}{|P(q)|} \quad (16)$$

where *q* is the query point, *H(q)* is the set of points in the query range *R*, *P(q)* is the set of all probed points. A higher VPR shows that more points in query range can be found when probe the same number of points.

C. Implementation Details

We use MPLSH as baseline to evaluate our method. We implement a MPLSH index structure modified from E2LSH. However, we only implement query-directed probing, because it is more similar to our method than step-wise probing. We also implement our method based on E2LSH. C++ programming language is used. The evaluation is done on a PC with one Intel dual-processor 2.4 GHz CPU and 2GB DRAM.

V. EEPERIMENTAL RESULTS

We use one hash table to index the test dataset in main memory. Every experiment repeats 10 times. The presented results are all average values. Three parameters, k, w and l must be set for constructing LSH index structure. Besides, query range R need be set to realize range queries. k is used to control the precision of the query result[1]. However, in section 4.2, we conclude that precision is not a valid metric for range queries. As a result, the only requirement for k is that k must remain unchanged in all experiments. In our experiment, the k is equal to 5. According to the conclusion in [3], w is set to 4 to optimize the performance. l is used to control the recall, but we can use only one hash table because we can get needed recall through multi-probe. In order to simulate the real situation, h_i is randomly selected but kept same for QRSP-MPLSH and MPLSH in every experiment.

A. Comparing Recall

We do two experiments to compare the recall of two methods. In the first experiment, we change the query range and compute the average recall for both methods. For QRSP-MPLSH, we use formula (14) to control the query quality and let N =10. For MPLSH, we use a fixed number of probe steps (4 for every query) to control the query quality. All other parameters are same for both methods. The experimental result is presented in Fig. 2. Two curves have similar shape, but our method can get higher recall at all query ranges. One reason is that our method can dynamically adjust the number of probe steps to avoid probing too few or to many candidate buckets. Another reason is that our probability model is sensitive to the query range and can generate a better probe sequence for range query than MPLSH.

In the second experiment, we compute the recall when probing top *N* (N from 1 to 15) buckets. The experimental result is shown in Fig. 3. Our method can get a higher recall than MPLSH when probing top N buckets. When N = 4, our method can get the recall 8 percentage points higher than MPLSH.

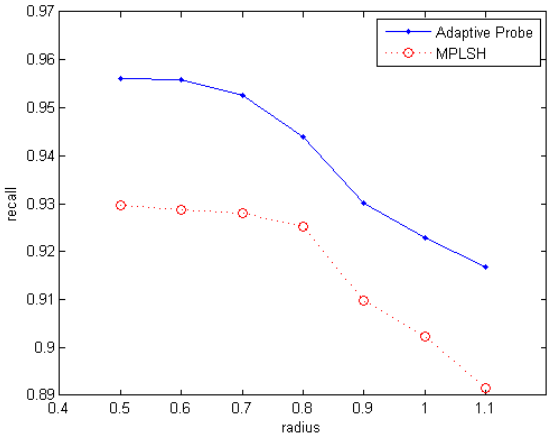

Figure 2. Recall vs. Query Range for QRSP-MPLSH and MPLSH. QRSP-MPLSH can get higher recall.

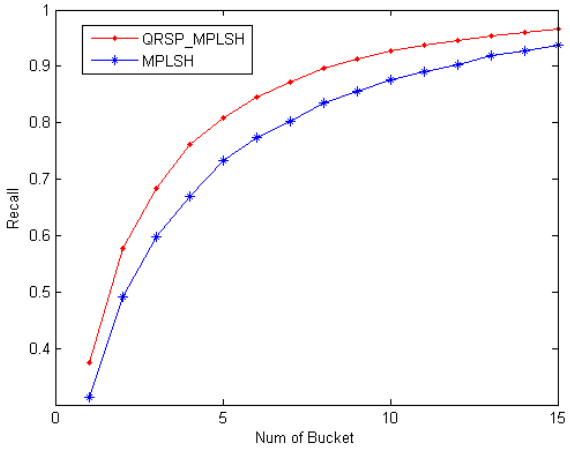

Figure 3. Recall vs. Number of Probed Bucket for QRSP-MPLSH and MPLSH. QRSP-MPLSH can get a higher recall when probe the same number of buckets.

B. Comparing Valid Probe Ratio

Although multi-probe method can get more valid points (e.g. points in the query range), it must also probe more invalid points (e.g. points out of the query range). In this situation, the ability to probe the least invalid points is

important. To evaluate this ability, Valid Probe Ratio (VPR) is introduced as an evaluation metric in Section 4.2. VPR can evaluate the efficiency of multi-probe. We compute the VPR at different recall when R = 0.5, 0.7, 0.9 and 1.1. For QRSP-MPLSH, we adjust N (in formula (14)) to get various recall. For MPLSH, we vary the number of probe steps to adjust recall. The experimental results are shown in Fig. 5.

As shown in Fig. 5, QRSP-MPLSH can get higher VPR at each recall than MPLSH. QRSP-MPLSH can improve VPR with 20% when R=0.5, recall=0.92 in Fig. 5. On the one hand, our probability model is more elaborate than MPLSH. On the other hand, our quality control method is more effective than MPLSH. Using query-adaptive threshold, our method can avoid probing too many buckets for some queries. For MPLSH, it must probe the same number of buckets for every query. Hence, MPLSH must probe more invalid points than our method. Based on the above two reasons, our method can get a better result than MPLSH in the experiment.

We can also observe that two curves in every plot are all degressive. This reports that multi-probe method's performance will become worse when it probes too many buckets.

In Fig. 5, the curve of QRSP-MPLSH covers a smaller range than that of MPLSH. At the left end, our initial recall is the same as the one of MPLSH. That is to say, the initial probed buckets of our method are same as MPLSH. At the right end, our maximal recall is smaller than that of MPLSH. This is because we let MPLSH probe all candidate buckets, and let our algorithm terminate when the probability is too small, i.e., our algorithm does not probe all candidate buckets.

C. Query-Adaptive and Query Range Sensitive

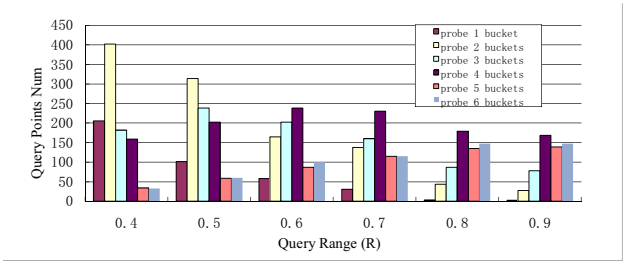

Figure 4. QRSP-MPLSH can adaptively adjust the number of probe steps for different query point and different R. However, MPLSH can only probe a fixed number of buckets.

In this experiment, we set N=10 and change the query range from 0.4 to 0.9. As shown in Fig. 4, different query points probe different number of buckets when R is fixed. For example, when R=0.4, there are 210 query points which probe one bucket, and there are 400 query points which probe two buckets. This proves that QRSP-MPLSH can adaptively choose how many buckets should be probed for a special query point. Furthermore, there are 210 query points which need to probe one bucket when R=0.4, but there are only 100 query points when R=0.5. This shows that QRSP-MPLSH can probe different number of buckets for some query points when R changes. Compared to QRSP-MPLSH, MPLSH does not have this kind of characteristic.

D. Selecting Threshold

We vary the threshold through changing N in the formula (14). If the value of N increases, the value of threshold decreases. Consequently, more buckets can be probed.

We generate the curve of Recall vs. N on a training dataset provided with the test dataset [11] in Fig. 6. It can be observed that the curve changes greatly when N is less than 10. The curve becomes smooth when N exceeds 10. This shows that the effect of the threshold becomes inconspicuous when the recall reaches a high value.

We also generate the curve of VPR vs. N in Fig. 7. The trend of the curve is opposite to that of Fig. 6. When N is smaller than 10, the curve is sharp. After N exceeds 10, the curve becomes smooth.

From Fig. 6 and Fig. 7, we get the conclusion that recall and VPR change oppositely. Based on this observation, we can assume that there is a value of N to balance recall and VPR. To get this value, we compute the product of recall and VPR. The curve in Fig. 8 shows the relationship between the product and N. There is an obvious peak value when N is 4. At this point, the recall is 0.8 and the VPR is 0.087. In practice, the recall may be more important than VPR. So, we can set N larger than the optimal value to get a higher recall. For example, we can set N = 10.

E. Comparing Computational Time

In Table 2, we compare the average query time of QRSP-MPLSH and MPLSH for 10000 query points. We vary the recall and the query range to get 9 pairs of records. According to the experiment result, the average query time of QRSP-MPLSH is shorter than that of MPLSH. The speedup can reach 1.175 at most. The minimal speedup is1.005 when r=0.5 and recall=0.65. It is very small. The reason is that the multi-probe sequences generated by two methods are same under this special condition. Except for this extreme case, the average speedup can get 1.100.

VI. Related Work

Compared to the basic LSH, multi-probe based methods extend the set of candidate buckets in each hash table. By probing neighbor buckets, the probability to find relevant neighbor points in a single hash table increases. Consequently, the number of hash tables can be reduced.

A. Multi-Probe LSH

Multi-Probe LSH [7] improves upon Entropy-Based LSH. A more efficient and accurate predict method based on a simple likelihood criterion is proposed. By this method, we can generate directly optimal probing sequence that is likely to contain more target points. The main contribution of MPLSH is as follows:

1. Propose a score function to guide the probe. The definition of the score function is the following:

$$score(\Delta) = \sum_{i=1}^{K} x_i(\delta_i)^2$$

2. Design two operations *shift* and exp*and* , which can generate the optimal probe sequence.

Compared with QRSP-MPLSH, the score function does not take query range into account. As a result, for range queries, our method can generate a better probe sequence than MPLSH. The experimental results show that our method can get a higher recall than MPLSH at the same test conditions. Furthermore, the experimental results prove that our method can probe fewer invalid points than MPLSH to get the same recall, because our probability model is sensitive to the query range. Besides, our method uses a query-adaptive threshold. Hence, our method can more effectively probe multiple buckets than MPLSH.

B. Posteriori Multi-Probe LSH

Posteriori Multi-Probe LSH [9] puts forward a more reliable posteriori model taking account some prior about the query and the searched objects. This prior knowledge helps to do a better quality control and more accurately select the most probable buckets. However, the probability model used in [9] does not take account the query range. After getting the probability of the candidate buckets, the candidate buckets are sorted by their probability value. Generally, to sort the probabilities, all the probabilities must be computed in advance. To avoid computing all the probabilities, three operations are proposed to generate the sorted probe sequence in an incremental style. In our algorithm, we use the similar operations.

VII. CONCLUSION

In this paper, we propose a novel query range sensitive probability model to predict which candidate bucket may contain the points in the query range with high probability. Because our model introduces the query range as a parameter, our method can generate a better probe sequence than other methods for range queries. To sort the candidate buckets' probabilities in a decreasing order, we construct a sorted matrix and use the similar operations as proposed in [9] to generate the optimal probe sequence. A query-adaptive threshold is used to control the probing procedure. For the different query point and the different query rang, there are different threshold values to control the probe steps. As a result, our method can reduce the invalid probes. For completely evaluating different multi-probe methods, Valid Probe Ratio (VPR) is used as an evaluation metric. A good multi-probe method should have higher VPR and recall at the same time. We implement QRSP-MPLSH and MPLSH, and do the experiments on an open dataset to compare both methods. The experimental results show that our method can generate a better probe sequence for range queries than MPLSH. Using the probe sequence generated by QRSP-MPLSH, the recall is improved by 8% and the VPR is improved by 20% at most. Furthermore, our method can get an average acceleration of 10% compared to MPLSH.

ACKNOWLEDGMENT

This work is supported by the National Nature Science Foundation of China (60802028); National High Technology and Research Development Program of China (863 Program, 2009AA01A403); Co-building Program of Beijing Municipal Education Commission; National key technology support program(2012BAH39B02).

REFERENCES

[1] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In Proc. of 25th Intl. Conf. on Very Large Data Bases(VLDB), pages 518–529, 1999.

[2] J. Buhler. Efficient large-scale sequence comparison by locality-sensitive hashing. Bioinformatics, 17:419–428, 2001.

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

[4] Andoni, A. and Indyk, P. E2lsh: Exact Euclidean locality sensitive hashing. http://web.mit.edu/andoni/www/LSH/. 2004.

[5] R. Panigrahy. Entropy based nearest neighbor search in high dimensions. In Proc. of annual ACM-SIAM symposium on Discrete algorithm, pages 1186–1195, 2006.

[6] Andoni, A. and Indyk, P. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In Proceedings of the Symposium on Foundations of Computer Science. 2006.

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

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

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

[10] H. Jegou, L. Amsaleg, C. Schmid, and P. Gros. Query-adaptative locality sensitive hashing. In International Conference on Acoustics, Speech, and Signal Processing. IEEE, 2008.

[11] http://corpus-texmex.irisa.fr/.

[12] Kulis, B. and Grauman, K. Kernelized locality-sensitive hashing for scalable image search. In Proceeding of 12th International Conference on Computer Vision. Pages 2130-2137, 2009.

[13] Y. Ke, R. Sukthankar, and L. Huston, "Efficient near-duplicate detection and sub-image retrieval," in ACM Conf. on Multimedia, 2004, pp. 869–876.

[14] G. Shakhnarovich, T. Darrell, and P. Indyk, Nearest-Neighbor Methods in Learning and Vision: Theory and Practice. MIT Press, Mar 2006, ch. 3.

[15] B. Matei, Y. Shan, H. Sawhney, Y. Tan, R. Kumar, D. Huber, and M. Hebert, "Rapid object indexing using locality sensitive hashing and joint 3D-signature space estimation," IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 28, no. 7, pp. 1111 – 1126, July 2006.

[16] Loïc Paulevé, Hervé Jégou, Laurent Amsaleg, Locality sensitive hashing: A comparison of hash function types and querying mechanisms, Pattern Recognition Letters, Volume 31, Issue 11, 1 August 2010, Pages 1348-1358

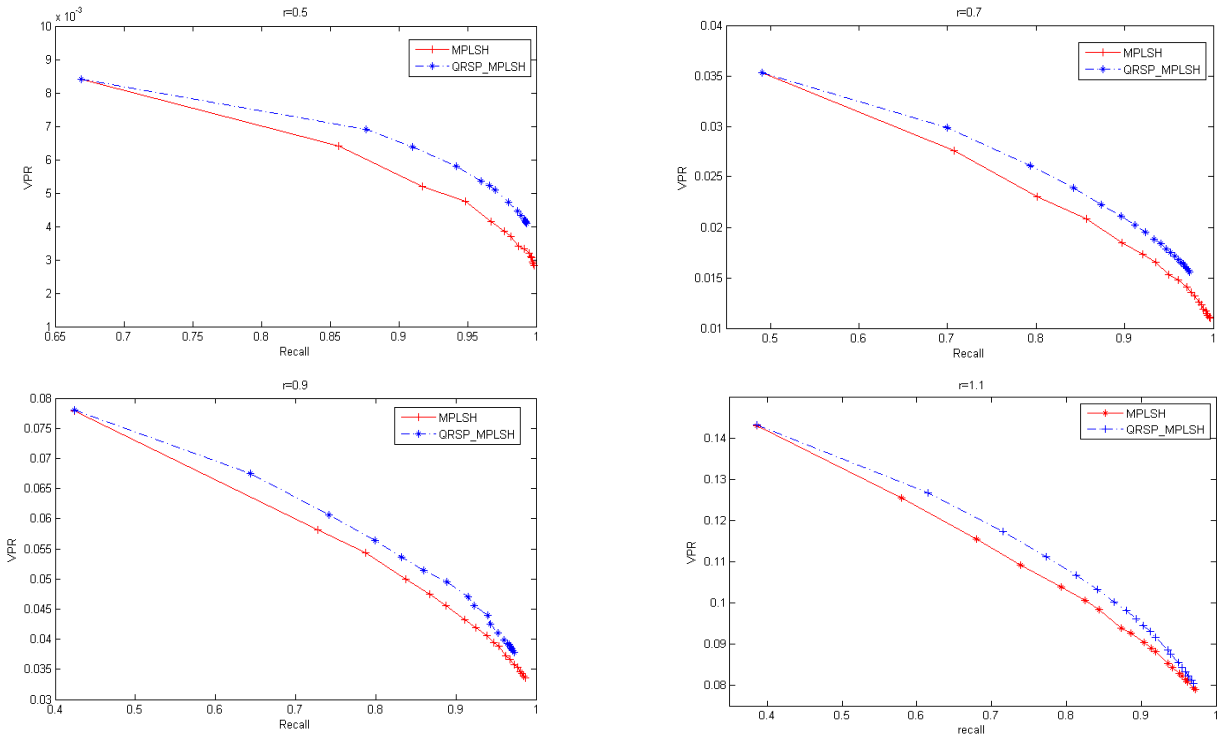

Figure 5. VPR vs. Recall for QRSP-MPLSH and MPLSH. QRSP-MPLSH can get a higher VPR for each recall.

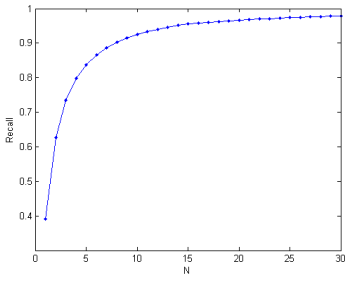

Figure 6. Recall vs. N for QRSP-MPLSH.

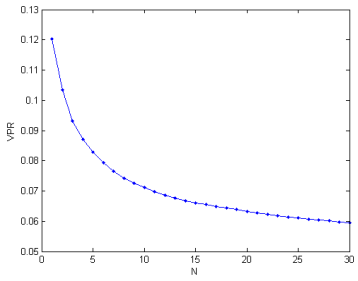

Figure 7. VPR vs. N for QRSP-MPLSH.

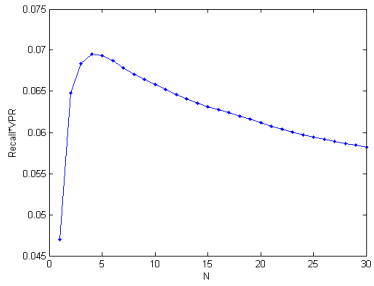

Figure 8. Product of Recall and VPR vs. N for QRSP-MPLSH.

TABLE 2. COMPARISON OF AVERAGE QUERY TIME (MEASURE:SECOND)

Recall	Method	r=0.5	Speedup	r=0.7	Speedup	r=0.9	Speedup
0.65	MPLSH QRSP	0.00200 0.00199	1.005	0.00266 0.00245	1.086	0.00470 0.00437	1.075
0.75	MPLSH QRSP	0.00253 0.00238	1.063	0.00331 0.00291	1.137	0.00540 0.00502	1.075
0.85	MPLSH QRSP	0.00355 0.00302	1.175	0.00401 0.00356	1.126	0.00596 0.00560	1.064