PM-LSH: A Fast and Accurate LSH Framework for High-Dimensional Approximate NN Search

Bolong Zheng[1], Xi Zhao[1], Lianggui Weng[1], Nguyen Quoc Viet Hung[2], Hang Liu[3], Christian S. Jensen[4]
[1]Huazhong University of Science and Technology [2]Griffith University
[3]Stevens Institute of Technology [4]Aalborg University
bolongzheng@hust.edu.cn, zhaoxi@hust.edu.cn, liangguiweng@hust.edu.cn, quocviethung.nguyen@griffith.edu.au, hangliu@stevens.edu, csj@cs.aau.dk

ABSTRACT

Nearest neighbor (NN) search in high-dimensional spaces is inherently computationally expensive due to the curse of dimensionality. As a well-known solution to approximate NN search, locality-sensitive hashing (LSH) is able to answer c-approximate NN (c-ANN) queries in sublinear time with constant probability. Existing LSH methods focus mainly on building hash bucket based indexing such that the candidate points can be retrieved quickly. However, existing coarse-grained structures fail to offer accurate distance estimation for candidate points, which translates into additional computational overhead when having to examine unnecessary points. This in turn reduces the performance of query processing. In contrast, we propose a fast and accurate LSH framework, called PM-LSH, that aims to compute the c-ANN query on large- scale, high-dimensional datasets. First, we adopt a simple yet effective PM-tree to index the data points. Second, we develop a tunable confidence interval to achieve accurate distance estimation and guarantee high result quality. Third, we propose an efficient algorithm on top of the PM-tree to improve the performance of computing c-ANN queries. Extensive experiments with real-world data offer evidence that PM-LSH is capable of outperforming existing proposals with respect to both efficiency and accuracy.

PVLDB Reference Format:
Bolong Zheng, Xi Zhao, Lianggui Weng, Nguyen Quoc Viet Hung, Hang Liu, Christian S. Jensen. PM-LSH: A Fast and Accurate LSH Framework for High-Dimensional Approximate NN Search. *PVLDB*, 13(5): 643-655, 2020.
DOI: https://doi.org/10.14778/3377369.3377374

1. INTRODUCTION

Nearest neighbor (NN) querying in high-dimensional spaces is classic functionality that is used in a wide variety of important applications, such as sequence matching [1], recommendation [8], similar-item retrieval [19], and de-duplication [24], to name but a few. Let $\mathcal{D}$ be a set of points in d-dimensional space $\mathbb{R}^d$. Given a query point q, an NN query returns a point o^* in $\mathcal{D}$ such that its Euclidean distance to q is the minimum among all points in $\mathcal{D}$.

While the exact NN query in low-dimensional space already has efficient solutions [5, 6], providing an efficient solution for large-scale datasets with high dimensionality remains a challenge, as both the query time and the space cost may increase exponentially with respect to the dimensionality. This phenomenon is called the "curse of dimensionality." Fortunately, it usually suffices to find an approximate nearest neighbor (ANN). For a given approximation ratio c ($c > 1$) and a query point q, a c-ANN query returns a point o whose distance to q is at most cr^*, where r^* is the distance between q and its exact NN o^*.

A widely-adopted locality-sensitive hashing (LSH) method enables computing c-ANN queries in sublinear time with constant probability. Generally, LSH maps the points in the dataset to buckets in hash tables by using a set of predefined hash functions that are designed to be locality-sensitive so that close points are hashed to the same bucket with high probability. A query is answered by examining the points that are hashed to the same bucket as the query point, or to similar buckets. Based on their main ideas, we classify the mainstream LSH methods into three categories: 1) Probing Sequence based (PS) approaches [20, 22, 23]; 2) Radius Enlarging based (RE) approaches [11, 17, 33]; and 3) Metric Indexing based (MI) approaches [31]. PS approaches use a carefully derived probing sequence to examine multiple hash buckets that are likely to contain the nearest neighbor of a query. RE approaches process a sequence of range queries by enlarging the query range repeatedly until a qualified point is found. In MI approaches, the points are transformed into a low-dimensional space, called the projected space. The coordinates of a point in the projected space are the point's hash values. MI approaches then use a metric index to organize the points such that the distance between two points in the projected space can be used to approximate the distance between them in the original space.

When evaluating the performance of LSH methods, many pertinent performance metrics for c-ANN search exist, including efficiency, accuracy, memory consumption, and preprocessing overhead. Among these, both *efficiency* and *accuracy* are important metrics since a desirable algorithm should return results as soon as possible with a quality that is as high as possible, while the memory consumption and preprocessing overhead must be tolerable in the setting of

Proceedings of the VLDB Endowment, Vol. 13, No. 5
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3377369.3377374

a commodity machine. The performance of LSH depends on two aspects: 1) the estimation of distances between the query point and candidate points; and 2) the probing order of buckets/points. It is proven [31] that the ratio of the projected distance over the original distance between any two points follows a χ^2 distribution. Therefore, if we are able to estimate the distance between two points accurately, we are able to find high quality candidates. In addition, a well-designed index structure is required to quickly locate high-quality candidates.

However, the existing LSH methods suffer from either inaccurate distance estimation or unnecessary point probing overhead. For instance, SRS [31] is the state-of-the-art algorithm that uses an R-tree to index the points in the projected space. By searching the R-tree, SRS is able to iteratively return the next nearest point to q. The problem is that finding the next exact NN in an R-tree generally causes additional computational overhead, while the next NN is not necessarily the best next candidate in the original space. Next, Multi-Probe [22] iteratively identifies the next hash bucket to be examined that has the least distance to q. However, most of the points in the identified buckets have to be probed due to poor estimation of the distance between q and the candidate point. Finally, QALSH [17] shares the same issue as Multi-Probe, and it uses a large number of hash functions that may incur high space consumption.

In this paper, we propose a fast and accurate LSH framework, called PM-LSH, for computing c-ANN queries on large-scale, high-dimensional datasets. The framework consists of three components, namely data partitioning, distance estimation, and point probing. First, we adopt the simple yet effective PM-tree [30] to index the points in the projected space. Second, in order to improve the distance estimation accuracy, we exploit the strong relationship between the original and projected distance of any two points, and we develop a tunable confidence interval on the projected distance w.r.t. a given original distance. Third, we propose an efficient algorithm to search the PM-tree with a sequence of range queries with increasingly large radius. PM-LSH is able to achieve both high efficiency and high accuracy when compared with the existing LSH methods.

The major contributions are summarized as follows:

- We present a unified interpretation of the existing mainstream LSH methods and thoroughly analyze the competitors in relation to our method.
- We propose a fast and accurate method called PM-LSH for large-scale, high-dimensional datasets. First, we use the PM-tree to index the points in the projected space. Second, we develop a tunable confidence interval for distance estimation. Third, we propose an efficient algorithm to search the PM-tree for computing c-ANN queries.
- We conduct an extensive performance study using real datasets that covers the state-of-the-art algorithms, which indicates that PM-LSH is efficient as well as accurate in terms of both the overall ratio and recall.

The rest of the paper is organized as follows. Section 2 presents the problem setting and preliminaries. Section 3 introduces a unified LSH framework and contrasts it w.r.t. the main competitors. Section 4 explains the construction and query algorithms of the PM-LSH, and Section 5 establishes its performance guarantees. Section 6 covers experimental studies that offer insight into the performance of the proposed PM-LSH and the main competitors. Section 7 reviews related work. Finally, Section 8 concludes the paper.

Table 1: Summary of Notations

Notation	Definition
$\mathcal{D}$	Dataset of points in $\mathbb{R}^d$
$n = \|\mathcal{D}\|$	Dataset cardinality
d	Dimension
o	A point in $\mathcal{D}$
o'	A point o in the projected space
c	Approximation ratio
$h(o)$, $h^*(o)$	Hash functions

2. PRELIMINARIES

We present the problem definition and basic idea of LSH. Frequently used notation is summarized in Table 1.

2.1 Problem Definition

We study the c-ANN and (c,k)-ANN queries. Let $\mathcal{D}$ be a set of points in d-dimensional space $\mathbb{R}^d$ with cardinality $|\mathcal{D}| = n$. Let $||o_1, o_2||$ denote the Euclidean distance between points $o_1, o_2 \in \mathcal{D}$.

Definition 1. ***c-ANN Query.*** *Assume a query point q and an approximation ratio $c > 1$, and let o^* be the exact nearest neighbor of q in $\mathcal{D}$. A c-approximate nearest neighbor query returns a point $o \in \mathcal{D}$ such that $||q, o|| \leq c \cdot ||q, o^*||$.*

We generalize the c-ANN query to the (c,k)-ANN query that returns k approximate nearest points.

Definition 2. ***(c,k)-ANN Query.*** *Assume we have a query point q, an approximation ratio $c > 1$, and a positive integer k. Let o_i^* be the i-th exact nearest neighbor of q in $\mathcal{D}$. A (c,k)-approximate nearest neighbor query returns a sequence of k points $\langle o_1, o_2, \ldots, o_k \rangle$ such that for each o_i, we have $||q, o_i|| \leq c \cdot ||q, o_i^*||$, $i \in [1,k]$.*

Example 1. *As shown in Fig. 1(a), query q has o_2 and o_{14} with distance $\sqrt{2}$ as its exact NNs. For a 2-ANN query, any point whose distance to q is within $2\sqrt{2}$ can be considered as a result, i.e., any object in the set $\{o_2, o_{14}, o_{12}, o_{13}, o_6, o_7\}$.*

2.2 Basic Locality Sensitive Hashing

We first introduce the LSH scheme, and then explain how to answer the (r,c)-ball cover ((r,c)-BC) and c-ANN queries using the basic LSH [3, 9].

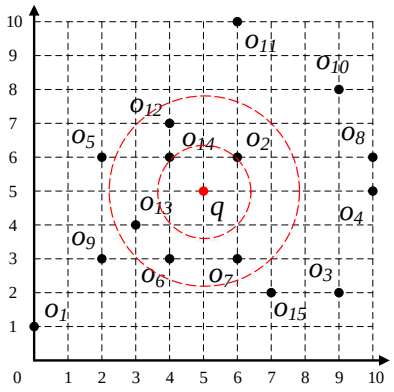
Hash Family. Given a distance r, an approximation ratio $c > 1$, probability values p_1 and p_2, where $p_1 > p_2$, a family $\mathcal{H} = \{h : \mathbb{R}^d \to U\}$ is called (r, cr, p_1, p_2)-locality sensitive, if for any $o_1, o_2 \in \mathbb{R}^d$, it satisfies both of the following conditions:

1. If $||o_1, o_2|| \leq r$ then $Pr[h(o_1) = h(o_2)] \geq p_1$
2. If $||o_1, o_2|| \geq cr$ then $Pr[h(o_1) = h(o_2)] \leq p_2$

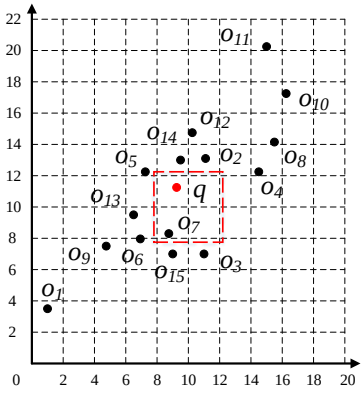
A well-adopted hash function is formally defined as follows:

$$h(o) = \lfloor \frac{\vec{a} \cdot \vec{o} + b}{w} \rfloor, \quad (1)$$

where $\vec{o}$ is the vector representation of a point $o \in \mathbb{R}^d$, $\vec{a}$ is a d-dimensional vector where each dimension is drawn independently from a p-stable distribution [9], b is a real number



(a) Original Space



(b) Projected Space

ID	(x,y)	h*(o)	ID	(x,y)	h*(o)
q	(5,5)	(9.5,11.5)	o_8	(10,6)	(15.4,14.2)
o_1	(0,1)	(0.9,3.7)	o_9	(2,3)	(4.7,7.5)
o_2	(6,6)	(11.4,13.4)	o_{10}	(9,8)	(16.2,17.4)
o_3	(9,2)	(10.8,7.2)	o_{11}	(6,10)	(15.0,20.2)
o_4	(10,5)	(14.5,12.5)	o_{12}	(4,7)	(10.3,14.7)
o_5	(2,6)	(7.4,12.6)	o_{13}	(3,4)	(6.6,9.4)
o_6	(4,3)	(6.7,7.9)	o_{14}	(4,6)	(9.4,13.0)
o_7	(6,3)	(8.7,8.3)	o_{15}	(7,2)	(8.8,6.8)

(c) Points and Hash Values

Figure 1: Running Example with $h_1(o) = \lfloor \frac{\vec{a_1} \cdot \vec{o}}{4} \rfloor$, $h_2(o) = \lfloor \frac{\vec{a_2} \cdot \vec{o} + 2}{4} \rfloor$ and $\vec{a_1} = [1.0, 0.9]$, $\vec{a_2} = [0.2, 1.7]$

uniformly drawn from $[0, w)$, and w is a user-specified constant. The 2-stable distribution is the normal distribution.

Formally, let $\tau = ||o_1, o_2||$, and let $f(\cdot)$ denote the normal probability distribution function (pdf). We then have:

$$p(\tau) = Pr[h(o_1) = h(o_2)] = \int_0^w \frac{1}{\tau} \cdot f(\frac{t}{\tau}) \cdot (1 - \frac{t}{w})\, dt \quad (2)$$

The intuition behind Eq. 2 is that, given a fixed w, the collision probability of two hash values $h(o_1)$ and $h(o_2)$ grows as the distance $||o_1, o_2||$ decreases. Therefore, $h(\cdot)$ in Eq. 1 is (r, cr, p_1, p_2)-sensitive with $p_1 = p(r)$ and $p_2 = p(cr)$.

(r, c)-BC Query. Before we consider how to answer the c-ANN query, we define an (r, c)-ball cover query that can be directly answered by (r, cr, p_1, p_2)-sensitive hash family.

Definition 3. ***(r, c)-BC Query.*** *Given a query point q, a distance threshold r, and an approximation ratio $c > 1$. Let $B(q, r)$ denote a ball centered at q with radius r. An (r, c)-ball cover query returns the following result:*

1. *If $B(q, r)$ covers at least one point in $\mathcal{D}$, it returns a point in $B(q, cr)$;*
2. *If $B(q, cr)$ covers no points in $\mathcal{D}$, it returns nothing.*

E2LSH [3] is a seminal solution that forms L hash tables and randomly chooses m hash functions for each hash table. By concatenating the m hash functions, a compound hash function $G(o) = (h_1(o), \ldots, h_m(o))$ is formed in each hash table, and each point $o \in \mathcal{D}$ is stored in a hash bucket based on $G(o)$. Given a query point q, E2LSH computes $G(q)$ and enumerates the points in the corresponding hash bucket. In all L hash tables, it examines at most $3L$ points and returns a point o if $||q, o|| \leq cr$. By setting $m = \log_{1/p_2} n$ and $L = 1/p_1^k$, the (r, c)-BC query can be answered correctly with at least constant probability.

From (r, c)-BC to c-ANN. It is easy to see that the ball cover query can be considered as a decision version of the approximate NN query. By processing a sequence of (r, c)-BC queries with $r = 1, c, c^2, \ldots, x$, once a point is returned, we take it as a result of the ANN query. Interestingly, as proven by [18], the ANN query can be answered with an approximation ratio c^2, i.e., c^2-ANN.

Example 2. *In the example in Fig. 1, we choose $m = 2$ hash functions $h_1(o) = \lfloor \frac{\vec{a_1} \cdot \vec{o}}{4} \rfloor$, $h_2(o) = \lfloor \frac{\vec{a_2} \cdot \vec{o} + 2}{4} \rfloor$ with $\vec{a_1} = [1.0, 0.9]$, $\vec{a_2} = [0.2, 1.7]$, $b_1 = 0$, $b_2 = 2$, and $w = 4$. For simplicity, we only construct $L = 1$ hash table. Figs. 1(b) and 1(c) show the coordinates of the objects in the projected space. To answer a $(1, 2)$-BC query with $r = 1$ and $c = 2$, we first compute $G(q) = (h_1(q), h_2(q)) = (2, 2)$. Then we search the hash bucket $(2, 2)$ that is indicated by a red rectangle, and the $(1, 2)$-BC query returns o_7. As o_{14} is the exact NN with $||q, o_{14}|| = \sqrt{2}$ and $||q, o_7|| = \sqrt{5} < 4 \times \sqrt{2}$, we have that o_7 is a result of the 4-ANN query of q.*

3. A UNIFIED INTERPRETATION OF LSH

We proceed to introduce the main competitors to our algorithm and give a unified interpretation.

3.1 Main Competitors

Probing Sequence Based (PS) Approaches. The representative PS methods include Multi-Probe [22, 23] and GQR [20] that use a carefully derived probing sequence to examine multiple hash buckets that are likely to contain the nearest neighbors of a query point. Unlike the basic LSH that builds L hash tables and checks only one hash bucket in each hash table, PS probes multiple nearby buckets in order to achieve higher recall with fewer hash tables. Given a query point q, PS adopts a "generate-to-probe" paradigm that iteratively generates the next hash bucket to be examined with the least distance to q in the remaining buckets. Note that although GQR is claimed to work for L2H [34] with binary code, we introduce it in order to better explain the intuition of PS.

Radius Enlarging Based (RE) Approaches. This category mainly includes the LSB-Tree [33], C2LSH [11], and QALSH [17]. These do not build multiple hash tables based on different radii. Generally, RE builds a hash table like the basic LSH and processes a sequence of (r, c)-BC queries by enlarging $r = 1, c, c^2, \ldots, x$ when a c-ANN query is issued. Suppose $r_i = c^i$ and $r_0 = 1$. It has been shown [11] that $h^{r_i}(\cdot) = \lfloor \frac{h(\cdot)}{r_i} \rfloor$ is (r_i, cr_i, p_1, p_2)-sensitive. Instead of building multiple hash tables with corresponding hash functions $h^{r_i}(\cdot)$ to handle (r_i, cr_i)-BC queries, RE adopts the smart idea of "virtual rehashing" to avoid unnecessary space. For the $(1, c)$-BC query, RE probes the hash bucket $h(q)$. For

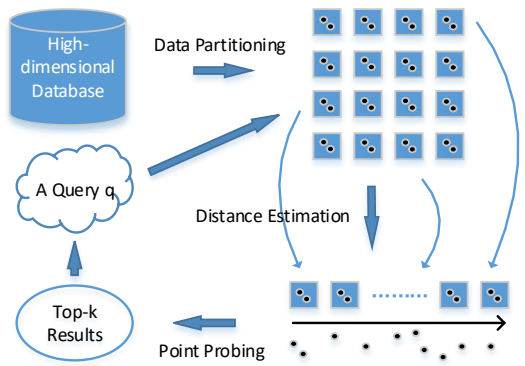


Figure 2: Unified LSH Framework

the remaining (r_i, cr_i)-BC queries, RE probes r_i^m hash buckets near $h(q)$ in the original hash table in the i-th iteration. Note that among these r_i^m buckets, r_{i-1}^m buckets were already examined in the last iteration. Interestingly, it is easy to see that the r_i^m hash buckets in the original hash table actually correspond to the hash bucket $h^{r_i}(q)$ in the hash table w.r.t. $h^{r_i}(\cdot)$.

Metric Indexing Based (MI) Approaches. SRS [31] is the state-of-the-art algorithm that projects the points from the original d-dimensional space into a lower m-dimensional projected space by using m hash functions. It utilizes an R-tree to index the points based on their hash values in the projected space. Specifically, SRS uses the Euclidean distance between two points in the projected space to approximate their distance in the original space. The intuition is that the points close to the query point q in the projected space are also likely close to q in the original space. SRS repeatedly calls an incSearch function that utilizes the R-tree to return the next nearest point to q in the projected space.

3.2 A Way of Probing

We proceed to introduce a unified interpretation of existing LSH methods as shown in Fig. 2, which consists of three components, namely data partitioning, distance estimation, and point probing.

Generally, $h(o)$ in Eq. 1 can be considered as a derivation of a family of locality sensitive hash functions:

$$h^*(o) = \vec{a} \cdot \vec{o} \tag{3}$$

By using $h^*(o)$, the points in the original space are mapped into a projected space, as shown in Figs. 1(a) and 1(b). Let $o' = [h_1^*(o), \ldots, h_m^*(o)]$ denote point o in the projected space. For any two points o_1 and o_2, let $r = ||o_1, o_2||$ and $r' = ||o_1', o_2'||$ denote the distance between o_1 and o_2 in the original and in the projected space, respectively. In addition, we let $\rho(o_1, o_2)$ denote an m-dimensional vector, where each dimension is the hash value difference between o_1 and o_2, i.e., $\rho_i = h_i^*(o_1) - h_i^*(o_2) = o_1'[i] - o_2'[i]$. It is easy to see that $r' = \sqrt{\sum_{i=1}^m \rho_i^2}$.

According to a property of a 2-stable distribution, for any d real numbers $o[1], \ldots, o[d]$, independent and identically distributed (i.i.d.) random variables $X_1, \ldots, X_d$ (corresponding to $\vec{a}$) following the 2-stable distribution, $\sum_i o[i] \cdot X_i$ has the same distribution as the variable $(\sum_{i=1}^d o[i]^2)^{1/2} \cdot X$, where X is a random variable with distribution $N(0, 1)$. For any two points o_1 and o_2, since $\rho = h^*(o_1) - h^*(o_2) = \vec{a} \cdot (\vec{o_1} - \vec{o_2})$, we know that ρ is a random variable with distribution $r \cdot X$. In other words, ρ has distribution $N(0, r^2)$, i.e., $\frac{\rho}{r} \sim N(0, 1)$.

LEMMA 1. *r'^2/r^2 follows the distribution $\chi^2(m)$.*

PROOF. If $Y_1, \ldots, Y_m$ are i.i.d. variables with $N(0, 1)$ then $\sum_{i=1}^m Y_i^2$ follows the χ^2 distribution with m degrees of freedom. Given m hash functions $h_1^*(\cdot), \ldots, h_m^*(\cdot)$, for any o_1 and o_2, we have $\rho_1, \ldots, \rho_m$. Thus, r'^2/r^2 follows the distribution $\chi^2(m)$. □

Data Partitioning. After mapping the points into the projected space by using hash functions, the existing LSH methods adopt the "divide-and-conquer" paradigm that partitions the projected space into subspaces. When a query is issued, the regions that are likely to contain the results are probed, and finally the results of these regions are combined and returned. Generally, there are two kinds of data partitioning approaches in the existing LSH methods:

(1) **Interval based Partitioning**. The basic LSH constructs hash buckets based on $G(o)$, and each bucket can be viewed as an m-dimensional hypercube with equal side lengths w. Most of the LSH methods belong to this class, including Multi-Probe, LSB-Tree, C2LSH, and QALSH. Specifically, an LSB-Tree assigns each hypercube a Z-order value and stores the values in a B-tree. In contrast, QALSH does not physically build hypercubes, but stores the values of $h^*(o)$ in a B$^+$-tree. When a query arrives, the length-w intervals are virtually formed on the B$^+$-tree.

(2) **Metric Space Partitioning**. SRS uses an R-tree to index all the points o' in the projected space such that an incremental kNN search is supported. For in-memory processing, it is also able to use a Cover Tree. In our proposed PM-LSH, we partition the projected space using a PM-tree so that efficient range querying can be supported.

Distance Estimation. In order to accurately estimate distances, two aspects are considered, i.e., the distance estimator and the estimation granularity.

(1) **Distance Estimator**. As we know that ρ has distribution $N(0, r^2)$. For any o_1 and o_2, $\rho(o_1, o_2) = [\rho_1, \ldots, \rho_m]$. We estimate the value of r by using r' as follows.

LEMMA 2. *$\hat{r} = \frac{r'}{\sqrt{m}}$ is an unbiased estimator of r.*

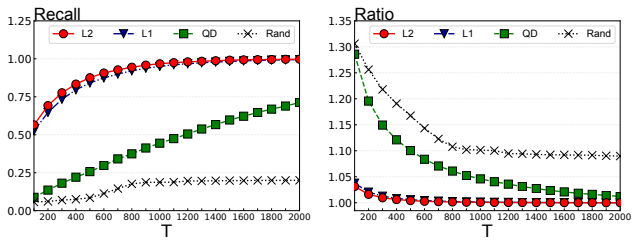
PROOF. Let $\hat{r}$ be the estimated value of r. We compute the expectation of r' as follows.

$$E[r'] = E[\sqrt{\sum_{i=1}^m \rho_i^2}] = \sqrt{\sum_{i=1}^m E[\rho_i^2]} = \sqrt{m}r$$

Therefore, we have $E[\hat{r}] = E[r']/\sqrt{m} = r$.

Alternatively, we provide a different yet interesting proof by using the maximum likelihood estimation (MLE) [15]. MLE is a procedure for finding the value of one or more parameters for a given statistic that maximizes the known likelihood distribution. As $Pr[\rho = \rho_i] = \frac{1}{\sqrt{2\pi}r} \exp(-\frac{\rho_i^2}{2r^2})$, the probability that the hash value difference $\rho(o_1, o_2)$ between o_1 and o_2 equals $[\rho_1, \ldots, \rho_m]$ is computed as follows.

$$\begin{aligned} Pr[\rho(o_1, o_2) &= [\rho_1, \ldots, \rho_m]] \\ &= f(\rho_1, \ldots, \rho_m | \mu = 0, \sigma = r) \\ &= \prod_{i=1}^m (\frac{1}{\sqrt{2\pi}r})^m \exp(-\frac{\sum_{i=1}^m \rho_i^2}{2r^2}) \end{aligned}$$



(a) Recall (b) Overall Ratio

Figure 3: Recall and Overall Ratio of Different Estimators

The objective of the maximum likelihood is to find an r such that the above probability is maximized. As $\ln f = -\frac{1}{2}m\ln(2\pi) - m\ln r - \frac{\sum \rho_i^2}{2r^2}$ and $\frac{\partial(\ln f)}{\partial r} = -\frac{m}{r} + \frac{\sum \rho_i^2}{r^3} = 0$, we have $\hat{r} = \sqrt{\frac{\sum_{i=1}^{m}\rho_i^2}{m}} = \frac{r'}{\sqrt{m}}$. □

To evaluate the performance of our estimator in Lemma 2, i.e., $L_2 = r'$ (the same as our estimator when m is fixed), we compare it with other distance estimators: L_1, QD [20], and Rand (assign a random value). We randomly sample a small dataset that contains 10K points from the *Trevi* dataset [21] and choose 100 points as query points. For each query point q, we first compute its exact 100NNs. Choosing $m = 15$ hash functions, we compute the distances in the projected space between q and all the points based on different estimators. Then, we choose the top-T points with smallest estimated distances (T varies from 100 to 2,000). For each q, we compare its exact 100NNs with the 100NNs from the T points. Finally, we compute the average *recall* and *overall ratio* (discussed in Section 6) of these 4 estimators. As shown in Fig. 3, we can see that our estimator has the best performance in terms of both the recall and overall ratio.

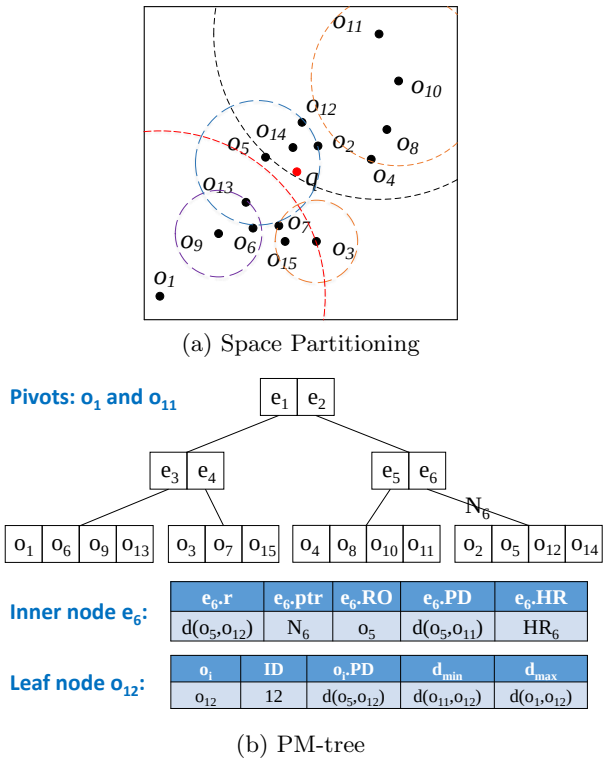
(2) **Estimation Granularity**. Distance estimation methods may use different granularities:

(i) **Bucket to Bucket**. The hash bucket based indexing methods, such as Multi-Probe, LSB-tree, and C2LSH, store points in hash buckets. When a query is issued, we first find its corresponding bucket and then decide which buckets to probe. Therefore, the quality of the distance estimation between buckets is affected by the bucket side length w.

(ii) **Point to Bucket**. QALSH is an improved version of C2LSH that stores points by a B$^+$-tree instead of using a hash table. When a query q arrives, the length-w intervals are conceptually built on the B$^+$-tree with q as the center. So the distance estimation can be considered as between point q and bucket intervals.

(iii) **Point to Point**. SRS uses the projected Euclidean distance between two points to estimate their original distance, which offers a finer precision than the previous two methods due to the fine granularity. Our PM-LSH also adopts this method.

Point Probing. Suppose we probe T points. In the hash bucket based indexing methods, such as Multi-Probe, LSB-tree, and C2LSH, we directly probe the points in the bucket, where the time cost is $O(T)$. The second approach is QALSH that searches the points in a B$^+$-tree. The time cost is $O(\log n + T)$. Unlike the previous two approaches, SRS indexes the points with an R-tree, and iteratively finds the next NN in the projected space. The time cost is $O(\log n \cdot T)$.



(a) Space Partitioning

(b) PM-tree

Figure 4: The Structure of PM-LSH

Our PM-LSH can be considered as a combination of the second and third approaches in that we build a PM-tree in the projected space and execute range queries to retrieve points.

4. THE PM-LSH FRAMEWORK

We proceed to present the details of the PM-LSH framework. As mentioned previously, the RE methods quickly probe the points stored in the hash buckets by enlarging the search radius, but suffer from inaccurate distance estimation due to a coarse-grained index structure, which translates into computational overhead when having to examine unnecessary points. In contrast, the MI methods index the points with an R-tree and iteratively return the next nearest point to q in the projected space. However, finding the next exact NN in an R-tree is also computationally costly, and the next NN is not necessarily the best next candidate in the original space. To achieve the best of both worlds, PM-LSH combines the ideas of the RE and MI methods, where we adopt the PM-tree instead of the R-tree to index the points in the projected space and execute a sequence of range queries with increasingly large radius such that both efficiency and accuracy are achieved.

Next, we briefly describe how to construct a PM-tree. Then, we analyze the cost models of the PM-tree and the R-tree to understand how the PM-tree performs better than the R-tree for the relevant range query workload. Finally, we present the details of the algorithms.

4.1 Building a PM-tree in the Projected Space

In the projected space, each o_i' w.r.t. $o_i \in \mathcal{D}$ is an m dimensional vector. For the paper to be self-contained, we

Table 2: Computation Cost (CC) of PM-tree and R-tree

Datasets	Audio	Cifar	MNIST	Trevi	NUS	GIST	Deep
PM-tree	38,182	35,210	56,670	34,281	201,448	739,720	964,451
R-tree	40,565	54,869	59,043	63,884	252,187	889,974	1,017,604
Reduction	6%	36%	4%	46%	20%	17%	5%

briefly explain how to build a PM-tree on all o_i's. Interested readers may refer to [30] for more details on the PM-tree.

Selecting Pivots. The PM-tree combines pivot mapping together with M-tree. Methods for selecting an optimal set of pivots have been studied extensively. For each set of pivots, a PM-tree region is the intersection of the M-tree hyper-spherical region and hyper-rings caused by the pivots. We choose a set of pivots with the aim of making the overall volume of the corresponding PM-tree region the smallest.

PM-tree Structure. The structure of a PM-tree is shown in Fig. 4. Since the PM-tree is an extension of the M-tree, it retains all the information of the M-tree. For an inner node e, it stores the covered radius $e.r$, a pointer to its covered sub-tree $e.ptr$, the center of the covered hyper-sphere $e.RO$, the distance $e.PD$ between $e.RO$ and its parent entry, and the smallest interval $e.HR$ covering the distances between the pivots and each of the point stored in leaves. For a leaf node o, it stores the point data, the ID of the point o, the distance $o.PD$ between o and its parent entry, and the minimum and the maximum distances to pivots.

Range Query Processing. To answer a range query, denoted as $\mathbf{range}(q, r)$, that returns all the points that locate in $B(q, r)$, the entries in the PM-tree are traversed in a depth-first fashion. When an inner node is accessed, we verify its pruning condition by using the triangle inequality. When a leaf entry is accessed, we insert the corresponding point into the result set if it is inside $B(q, r)$.

EXAMPLE 3. *As shown in Fig. 4, we choose o_1 and o_{11} as pivots, and partition the space by using the ball partitioning, as shown in Fig. 4(a). The inner nodes $e_1, e_2, \cdots, e_6$ contain the points inside a hyper-sphere region, whose center and radius are saved as the part of an entry. When a range query $\mathbf{range}(q, 2)$ is issued, we check the pruning conditions when accessing the inner nodes. Here only e_4 and e_6 are checked. Finally, we return $\{o_{14}\}$ as the result.*

4.2 Cost Models of the PM-tree vs the R-tree

To compare the performance of the PM-tree and the R-tree, we adopt a node-based cost model [7] to examine how the PM-tree performs compared to the R-tree from a theoretical point of view.

In this cost model, a concept called distance distribution of a dataset $\mathcal{D}$ is computed as follows.

$$F(x) = Pr[||o_i, o_j|| \leq x], \tag{4}$$

where $o_i, o_j \in \mathcal{D}$. In addition, for each dataset used in our experiments, we compute its "homogeneity of viewpoints" (HV), which is shown in Table 3. HV evaluates the homogeneity of the distance distributions of the data points. Let $F_o(x)$ denote the distribution of the distances between all points to point o. Given two points o_1 and o_2, a higher HV means that o_1 and o_2 are more likely to have similar distance distributions $F_{o_1}(x)$ and $F_{o_2}(x)$. The HV values of all the datasets are no smaller than 0.9, which enables us to approximate their distance distributions when estimating the cost models of the two trees.

Cost Model of the PM-tree. Consider a range query $\mathbf{range}(q, r_q)$. Assume that a PM-tree has s pivot points $p_1, \cdots, p_s$. An inner node e is accessed iff the follow conditions are satisfied:

$$\begin{cases} ||q, e.RO|| \leq e.r + r_q \\ \wedge_{i=1}^{s}\{||q, p_i|| - r_q \leq e.HR[i].max\} \\ \wedge_{i=1}^{s}\{||q, p_i|| + r_q \geq e.HR[i].min\} \end{cases} \tag{5}$$

Therefore, the probability of e being accessed can be computed as follows.

$$\begin{aligned} Pr[e] = & F(e.r + r_q) \\ & \cdot \prod_{i=1}^{s}[F(e.HR[i].max + r_q) \\ & - F(e.HR[i].min - r_q)] \end{aligned} \tag{6}$$

Assume that there are N nodes in the PM-tree. The number of distance computations (computation cost) is estimated by considering the probability that a node is accessed multiplied by its number of entries $N(e)$, thus obtaining the number of distance computations as follows.

$$\mathbf{CC}(\mathbf{range}(q, r_q)) = \sum_{i=1}^{N} N(e_i) \cdot Pr[e_i] \tag{7}$$

Cost Model of the R-tree. For each node e of an m-dimensional R-tree, we denote its minimum bounding rectangle as $MBR(e) = [l_1, u_1] \times \cdots \times [l_m, u_m]$. Given a range query $\mathbf{range}(q, r_q)$, the condition of e being accessed is that $B(q, r_q)$ intersects with $MBR(e)$. Since it is hard to quantify the probability that a ball intersects with a high-dimensional rectangle, we substitute an isochoric hyper-cube for the ball. Specifically, an m-dimensional ball with radius r_q can be substituted by a hyper-cube with the length of sides $l = \sqrt[m]{\frac{2\pi^{m/2}}{m\Gamma(m/2)}} r_q$ [18]. We also denote the data distribution of dataset $\mathcal{D}$ on the i-th dimension as follows.

$$G_i(x) = Pr[X_i \leq x], \tag{8}$$

where X_i is the i-th dimension of a random point in $\mathcal{D}$. Similarly, we let N be the number of nodes in the R-tree and let $N(e_i)$ be the number of entries in node e_i. We obtain the number of distance computations as follows (details are omitted for brevity).

$$\mathbf{CC}(\mathbf{range}(q, r_q)) = \sum_{i=1}^{N} N(e_i) \cdot \prod_{i=1}^{m}[G_i(u_i + l) - G_i(l_i - l)] \tag{9}$$

Comparison of the PM-tree and the R-tree. In order to compare the computation costs for the two trees, we construct PM-trees and R-trees for the points in all the datasets

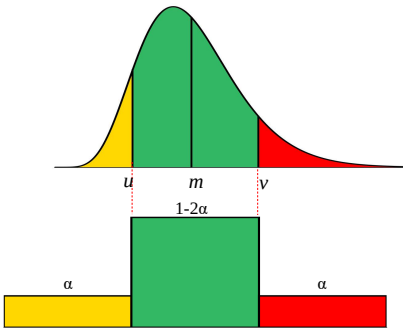


Figure 5: A Confidence Interval

(introduced in Table 3) after transforming them into the projected space. We choose $m = 15$ hash functions and set the maximum number of entries per node to 16. For each dataset, we choose the same range r to estimate the cost of computing a range query. The value of r is chosen to return approximately the nearest 8% of all points, since these points usually suffice to return a c-ANN result. The estimated computation costs are computed based on Eqs. 7 and 9, and the results are presented in Table 2. We can see that using the PM-tree reduces the number of distance computations by about $5\% - 46\%$ for the different datasets. This observation offers evidence that the PM-tree has better performance than the R-tree in our setting.

4.3 Tunable Confidence Interval

Based on Lemma 2, we further estimate the confidence interval of r' between o_1 and o_2 for a given $r = ||o_1, o_2||$.

LEMMA 3. *Given two points o_1 and o_2, we have:*

- ***P1**: The probability that $r' < r\sqrt{\chi^2_{1-\alpha}(m)}$ is α*
- ***P2**: The probability that $r' > r\sqrt{\chi^2_{\alpha}(m)}$ is α*

Here, $\chi^2_\alpha(m)$ is the upper quantile of a χ^2 distribution with m degrees of freedom, where

$$\int_{\chi^2_\alpha(m)}^{+\infty} f(x;m)dx = \alpha,$$

and $f(x;m)$ is the probability density function of a χ^2 distribution with m degrees of freedom.

PROOF. From Lemmas 1 and 2, we know $\frac{r'^2}{r^2} \sim \chi^2(m)$. Constructing a confidence interval $I = [u, v]$ for $\frac{r'^2}{r^2}$ requires that the probability that $\frac{r'^2}{r^2}$ falls into I is $1 - 2\alpha$ for any given α. A standard approach is to select u and v that make $Pr[\frac{r'^2}{r^2} < u] = \alpha$, i.e., $Pr[\frac{r'^2}{r^2} > u] = 1 - \alpha$, and $Pr[\frac{r'^2}{r^2} > v] = \alpha$. Further, $\int_u^{+\infty} f(x;m)dx = 1 - \alpha$ and $\int_v^{+\infty} f(x;m)dx = \alpha$. According to the definition of upper quantile, we have $u = \chi^2_{1-\alpha}(m)$ and $v = \chi^2_\alpha(m)$. The confidence interval and its corresponding probability are shown in Fig. 5. □

According to Lemma 3, we establish a strong relationship between an original distance and the confidence interval of a projected distance, which can be used to answer (r, c)-BC and c-ANN queries.

4.4 The (r, c)-BC Query

An (r, c)-BC query can be computed directly by Algorithm 1. Given a query q and m hash functions, we compute the hash value $q' = (h^*_1(q), \ldots, h^*_m(q))$ and use the PM-tree to answer a range query $\textbf{range}(q', tr)$, where t is a parameter that guarantees that a point inside $B(q, r)$ in the original space will fall into $B(q', tr)$ in the projected space with a constant probability. Then we collect the result of the range query into a candidate set C.

According to Lemma 4, to be introduced in Section 5, the correctness of the (r, c)-BC query can be guaranteed. In other words, by properly choosing a parameter β, we examine a sufficient number of βn candidate points, and the following two situations will hold with a constant probability.

- If the total number of points in C exceeds βn, there must be at least a point from C inside $B(q, cr)$.
- If there is no point in C inside $B(q, cr)$, there exists no point in $\mathcal{D}$ inside $B(q, r)$.

Therefore, we can correctly answer an (r, c)-BC query by processing a range query using the PM-tree. In Section 5, we consider how to set the parameters of t and β.

Algorithm 1: (r, c)-BC Query

```
Input: A query point q and parameters β, n, t, c, r
Output: A point p in B(q, cr) or nothing
1 Compute q' = (h*_1(q), ..., h*_m(q));
2 Initialize a candidate set C ← the results of a range
    query q' with radius t · r on the PM-tree;
3 if |C| ≥ βn + 1 then
4     return p in C that is closest to q;
5 else
6     if |{p | p ∈ C ∧ ||p, q|| ≤ c · r}| ≥ 1 then
7         return p in C that is closest to q;
8     else
9         return ∅;
```

4.5 The (c, k)-ANN Query

Answering a c-ANN query is more complicated than answering an (r, c)-BC query since we do not know the distance $||q, o^*||$ in advance. In order to answer a (c, k)-ANN query with a constant probability, we must ensure that we access enough points, i.e., at least βn points. Therefore, we have to enlarge the search radius in the projected space when fewer than βn points are found until k points inside $B(q, cr)$ have been obtained.

The details of computing a (c, k)-ANN query can be found in Algorithm 2. Most of the steps are almost the same as Algorithm 1. The difference is that when both termination conditions (Line 4 and Line 8) are violated, another range query with a larger radius is required.

Selecting the Radius r of a Range Query. As executing multiple range queries is time consuming, it is attractive to reduce the number of iteration in the while-loop. Intuitively, we hope to find a "magic" r_{min} such that the process terminates quickly. An ideal r_{min} must yield a number of points inside $B(q', tr_{min})$ that exceeds $\beta n + k$ such that Algorithm 2 is able to terminate after processing the range

query $B(q', tr_{min})$. In addition, to avoid returning a large number of unnecessary points, which also is costly, the number of points inside $B(q', tr_{min}/c)$ should be less than $\beta n+k$. Otherwise, the range query $B(q', tr_{min}/c)$ with smaller radius is able to return enough points.

As the r_{min} can be selected from a relatively large range, we design a selection scheme as follows. Suppose that we have obtained the distance distribution $F(x)$ of all datasets. Due to a good HV value, the distance distribution of a query point can be estimated by the dataset. Then we can find a suitable r that satisfies $n \cdot F(r) = \beta n+k$, which implies that $\beta n+k$ points locate in $B(q, r)$ on average. However, to avoid the case where the number of points in $B(q, r)$ exceeds $\beta n+k$, we choose an r_{min} slightly smaller than r. As the choice of r_{min} is not unique and the selection range is relatively large, and since the performance is not strongly dependent on it, the effect of the estimation is expected to be small.

EXAMPLE 4. *Setting $\beta n = 4$, we need to retrieve at least 5 points for a $(2, 1)$-ANN query. Initially, we set $r_{min} = r' = 2$. As explained in Example 3, o_{14} is returned. As the number of returned points is below 5, we set $r' = 4$. In this round, only the subtree of e_5 can be discarded, and we check the points in e_3, e_4, and e_6 and obtain $\{o_2, o_5, o_7, o_{12}, o_{13}, o_{14}\}$. The number of returned points is 6, and the process terminates. Finally, we return the $(2, 1)$-ANN result o_{14}.*

Algorithm 2: (c, k)-ANN Query

Input: A query point q, and parameters r_{min}, β, n, t, c, k
Output: A point p

```
Initialize a candidate set C ← ∅ and r ← r_min;
Compute q' = (h*_1(q), ..., h*_m(q));
while true do
    if |{p | p ∈ C ∧ ||p, q|| ≤ c · r}| ≥ k then
        return top-k points that are closest to q in C;
    Initialize a range query q' with radius t · r on the
     PM-tree;
    while |C| < βn + k do
        Find a node in B(q', t · r) on the PM-tree;
        C ← C ∪ {the points in the node};
    if |C| ≥ βn + k then
        return top-k points that are closest to q in C;
    r ← c · r;
```

5. THEORETICAL ANALYSIS

5.1 Quality Guarantee

In Algorithms 1 and 2, we execute a range query on the PM-tree with a radius tr in the projected space. Therefore, we have to compare the projected distances of candidate points to q with tr. Specifically, two types of points need to be discussed, true positives (the points inside $B(q, r)$) and false positives (the points outside $B(q, cr)$).

LEMMA 4. *Given a query q, by setting probabilities α_1 and α_2, and t that satisfy the following Eq. 10:*

$$\begin{cases} t^2 = \chi^2_{\alpha_1}(m) \\ t^2 = c^2\chi^2_{1-\alpha_2}(m) \end{cases} \tag{10}$$

We have:

- ***E1:** If a point o exists inside $B(q, r)$, its projected distance to q is smaller than tr.*
- ***E2:** There are fewer than βn ($\beta > \alpha_2$) points outside $B(q, cr)$ whose projected distances to q are smaller than tr.*

The probability that E1 occurs is at least $1 - \alpha_1$, and the probability that E2 occurs is at least $1 - \frac{\alpha_2}{\beta}$.

PROOF. Given a point $o \in B(q, r)$, let $r_o = ||o, q|| \leq r$ and $r'_o = ||o', q'||$ be the original and projected distances to q, respectively. By setting $t = \sqrt{\chi^2_{\alpha_1}(m)}$, according to the Lemma 3, we have $Pr[r'_o > r_o\sqrt{\chi^2_{\alpha_1}(m)}] = Pr[r'_o > tr_o] = \alpha_1$. Since $r_o \leq r$, $Pr[r'_o > tr]$ is at most α_1. Therefore, we know that $Pr[E1] = Pr[r'_o \leq tr] > 1 - \alpha_1$. Likewise, given a point $o \notin B(q, cr)$, let $r_o = ||o, q|| > cr$ and $r'_o = ||o', q'||$ be the original and projected distances to q, respectively. By setting $t = c\sqrt{\chi^2_{1-\alpha_2}(m)}$, according to the Lemma 3, we have $Pr[r'_o < r_o\sqrt{\chi^2_{1-\alpha_2}(m)}] = Pr[r'_o < t\frac{r_o}{c}] = \alpha_2$. Since $\frac{r_o}{c} > r$, $Pr[r'_o < tr]$ is at most α_2. Therefore, by using Markov's inequality, we have $Pr[E2] > 1 - \frac{\alpha_2}{\beta}$. □

Note that if $E1$ and $E2$ hold at the same time, then Algorithm 1 is correct for solving the (r, c)-BC query.

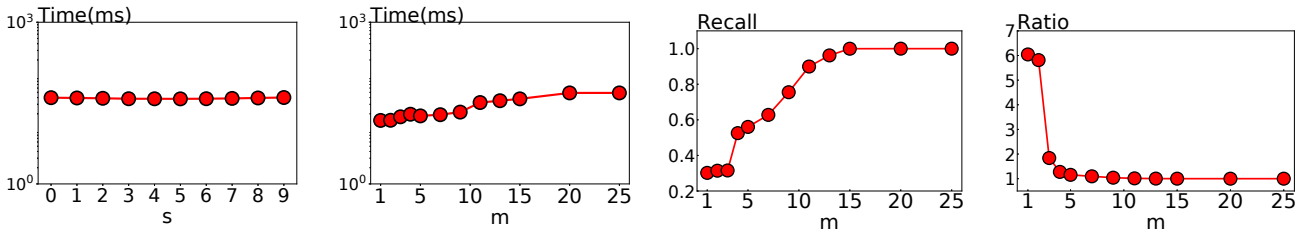
LEMMA 5. *Algorithm 1 answers an (r, c)-BC query with at least a constant probability.*

PROOF. Let $m = O(1)$. If α_1 is a constant, α_2 is also a constant due to Eq. 10. By setting $\beta = 2\alpha_2$, the lower bound probabilities of $E1$ and $E2$, i.e., $1 - \alpha_1$ and $1 - \frac{\alpha_2}{\beta}$, will also be constant. Therefore, we can guarantee that $E1$ and $E2$ hold at the same time with at least a constant probability. Thus, if we access at least $\beta n + 1$ points with projected distances smaller than tR to q, due to $E2$, there are at most βn points outside $B(q, cr)$, and we thus obtain at least one point inside $B(q, cr)$. On the other hand, if we access no more than $\beta n + 1$ points with projected distances smaller than tR to q, the correctness of $E2$ is not guaranteed. Therefore, it is safe to return either nothing or the points whose distances to q are at most cr for an (r, c)-BC query. □

As a typical setting in the LSH methods, we choose parameters that satisfy $Pr[E1] = 1 - 1/e$ and $Pr[E2] = 1/2$ (Of course, we can choose other parameters to achieve a more accurate result). Therefore, we have $\alpha_1 = 1/e$ and $t = \sqrt{\chi^2_{\alpha_1}(m)}$. Based on Eq. 10, both α_2 and β can be determined easily.

THEOREM 1. *Algorithm 2 returns a c^2-ANN with probability at least $1/2 - 1/e$.*

PROOF. Due to Lemma 5, we find that $E1$ and $E2$ can hold at same time with probability at least $1/2 - 1/e$ under such parameters. Now we show that when $E1$ and $E2$ hold, the output of Algorithm 2 is c^2-approximate. We denote the set of points whose projected distances to q are smaller than



(a) Time (b) Time (c) Recall (d) OverRatio

Figure 6: Performance of PM-LSH when Varying s and m

Table 3: Datasets

Dataset	n ($\times 10^3$)	d	HV	RC	LID
Audio	54	192	0.9273	2.97	5.6
Deep	1,000	256	0.9393	1.96	12.1
NUS	269	500	0.9995	1.67	24.5
MNIST	60	784	0.9531	2.38	6.5
GIST	983	960	0.9670	1.94	18.9
Cifar	50	1,024	0.9457	1.97	9.0
Trevi	100	4,096	0.9432	2.95	9.2

tr as $C(r)$. When enlarging $r = 1, c, c^2, \cdots$, there must exist a radius r_{opt} that makes $|C(r_{opt})| \geq 1+\beta n$ and $|C(r_{opt}/c)| < 1+\beta n$ hold. Then, if $r^* = ||o^*, q|| \leq r_{opt}/c$, its projected distance to q is smaller than tr_{opt}/c according to $E1$, we must have found it in $C(r_{opt})$ due to $C(r_{opt}) \supset C(r_{opt}/c)$, Algorithm 2 returns the exact NN; if $r = ||o^*, q|| > r_{opt}/c$, according to $E2$, there is at least a point in $C(r_{opt})$ whose distance to q is at most cr_{opt}. Therefore, we return a point whose distance to q is smaller than $c^2 r^*$. □

5.2 Algorithm Analysis

In PM-LSH, if we choose a large m, it will be costly to process a sequence of range queries in the projected space. So we consider m as a constant and fix its value at 15 in all experiments.

THEOREM 2. *PM-LSH has space cost $O(n)$ and time cost $O(\log n + \beta n)$, where β is much smaller than 1.*

PROOF. The space consumption is due mainly to the PM-tree, which has n items. Each item consumes $m+O(1)$ space, so the overall space consumption is $O(n)$ as $m = O(1)$. The query time cost comes from two parts: 1) finding candidate points in the PM-tree; and 2) verifying the real distances of candidate points to q. The former has cost $O(\log n)$ and the latter has cost $O(\beta n)$ when d is considered as a constant. Therefore, the total query time is $O(\log n + \beta n)$. □

6. EXPERIMENTS

6.1 Experimental Settings

All the algorithms are implemented in C++ compiled with the O3 optimization. All experiments are run on a Linux machine with an Intel 3.4GHz CPU and 32GB memory.

Datasets and Query Sets. We use seven real datasets: *Audio*, *Deep*, *NUS*, *MNIST*, *GIST*, *Cifar*, *Trevi*, which are used widely in existing work [11, 17, 20, 21, 31] on LSH. Table 3 reports the dimensionality and key statistics of the datasets: *Homogeneity of Viewpoints* (HV [7]), *Relative Contrast* (RC [16]), and *Local Intrinsic Dimensionality* (LID [2]). HV evaluates the homogeneity of the distance distributions of the data points. A higher HV means that the points are more likely to have similar distance distributions. RC computes the ratio of the mean distance over the NN distance for the data points. LID computes the local intrinsic dimensionality. A small RC value and a large LID value imply that it is challenging to compute NN results for the dataset. As queries, we randomly select 200 points from each dataset and repeat each experiment 20 times. We vary the value of k in $\{1, 10, 20, \ldots, 100\}$ and set the default value to 50. We vary the value of c in $\{1.1, 1.2, \ldots, 2.0\}$ and set the default value to 1.5.

Competing Algorithms. We compare PM-LSH with the competitors mentioned in Section 3, i.e., ① Multi-Probe, ② QALSH, and ③ SRS. In addition to these competitors, in order to study the advantages of the PM-tree over the R-tree, we index the points in the projected space with an R-tree instead of a PM-tree to see how PM-LSH then performs. We call this method ④ R-LSH. Moreover, we consider a linear scan algorithm called ⑤ LScan that randomly selects a portion of points (default 70%) and returns the top-k points with the smallest distances to the query.

Parameter Settings. We choose $m = 15$ hash functions for all the algorithms except QALSH and Multi-Probe. For PM-LSH, we set the number of pivots $s = 5$ and $\alpha_1 = 1/e$, so $\alpha_2 = 0.1405$ and $\beta = 0.2809$ are obtained according to Eq. 10, and r_{min} is determined according to description in the previous section. For QALSH, the false-positive percentage $\beta = 100/n$, and the error probability $\delta = 1/e$. For SRS, the threshold of its early-termination condition $p'_\tau = 0.8107$, and the maximum percentage of points accessed in the projected space is $T = 0.4010$ when $c = 1.5$.

Evaluation Metrics. We adopt three metrics to compare the performance of the algorithms: query time (ms), overall ratio, and recall, where the query time evaluates the algorithm efficiency, and the overall ratio and recall evaluate result quality. For a query q, we denote the result of a (c, k)-ANN query by $R = \{o_1, o_2, \cdots, o_k\}$. Let $R^* = \{o_1^*, o_2^*, \cdots, o_k^*\}$ be the exact kNNs. The overall ratio and recall are computed as follows.

$$OverallRatio = \frac{1}{k}\sum_{i=1}^{k}\frac{||q, o_i||}{||q, o_i^*||} \tag{11}$$

$$Recall = \frac{|R \cap R^*|}{|R^*|} \tag{12}$$

Table 4: Performance Overview

		PM-LSH	SRS	QALSH	Multi-Probe	R-LSH	LScan
Audio	Query Time (ms)	**13.5**	15.3	22.5	15.3	14.2	19.6
	Overall Ratio	**1.0014**	1.0025	1.0043	1.0242	1.0019	1.0073
	Recall	**0.9662**	0.9126	0.9003	0.8669	0.9633	0.6839
MNIST	Query Time (ms)	**12.3**	18.4	24.7	19.1	16.2	60.3
	Overall Ratio	**1.0076**	1.0101	1.0085	1.0103	1.0095	1.0276
	Recall	**0.8857**	0.8514	0.8655	0.8502	0.8705	0.7073
NUS	Query Time (ms)	**125.7**	142.1	133.2	125.9	129.6	176.8
	Overall Ratio	**1.0009**	1.0015	1.0027	1.0025	1.0011	1.0053
	Recall	**0.9257**	0.9247	0.8677	0.8782	0.9214	0.7057
Trevi	Query Time (ms)	**37.2**	47.9	145.5	239.3	63.9	57.68
	Overall Ratio	**1.0004**	1.0015	1.0029	1.0057	1.0044	1.0084
	Recall	**0.9961**	0.9342	0.8240	0.8534	0.9568	0.7103
Cifar	Query Time (ms)	**11.6**	16.1	38.3	26.8	35.6	58.2
	Overall Ratio	**1.0009**	1.0025	1.0057	1.0038	1.0056	1.0125
	Recall	**0.9746**	0.9624	0.7917	0.8011	0.9610	0.7081
GIST	Query Time (ms)	**398.7**	452.5	627.7	782.9	425.3	1528.3
	Overall Ratio	**1.0047**	1.0049	1.0037	1.0053	1.0059	1.0076
	Recall	**0.8436**	0.8145	0.8534	0.8122	0.8098	0.7023
Deep	Query Time (ms)	**227.8**	252.9	458.2	401.4	457.5	507.5
	Overall Ratio	**1.0037**	1.0077	1.0124	1.0112	1.0152	1.0145
	Recall	**0.8816**	0.8894	0.646	0.8118	0.8801	0.6938

6.2 Performance Evaluation

To evaluate the performance of PM-LSH, we first conduct an evaluation to determine parameter settings. Then, we compare the performances of all the algorithms with default parameter settings on all the datasets. Finally, we compare the algorithms by studying the changes of the overall ratio and recall under different the query times.

Parameter Study on PM-LSH. We discuss two parameters that may affect the performance of PM-LSH, i.e., the number of pivots s and the number of hash functions m. Here, we only show results from the *Trevi* dataset. It is easy to see that s only affects the query time. The overall ratio and recall will not change when we vary the value of s. As we can see from the Fig. 6(a), when s changes, the query time remains steady, which indicates that PM-LSH is largely unaffected by different settings for s. When using a larger number of pivots, we have a higher chance to prune subtrees in the PM-tree. However, the cost of checking on the pruning condition also increases. In conclusion, we set $s = 5$.

As shown in Fig. 6, when the value of m increases, we obtain a higher overall ratio and recall, but the query time also increases. The higher quality occurs because a larger m can lead to more accurate distance estimation. However, the average cost to retrieve a point from the PM-tree also increases. Taking both efficiency and accuracy into consideration, we set $m = 15$.

When comparing PM-LSH with R-LSH, we observe in all the experiments that PM-LSH outperforms R-LSH on all metrics, which confirms the expected superiority of the PM-tree over the R-tree.

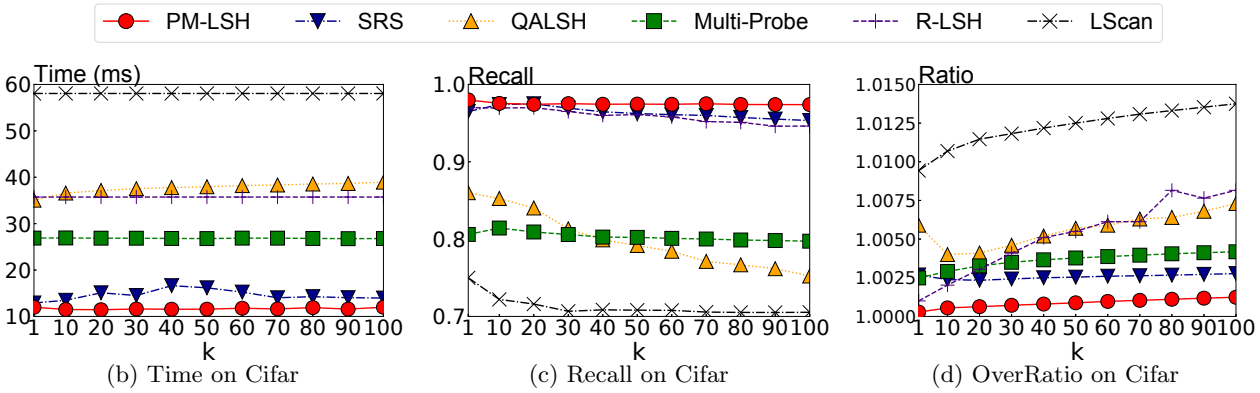
Performance Overview. To compare all the algorithms with default parameter settings, we report the query time (ms), overall ratio, and recall on all datasets in Table 4. PM-LSH is more efficient than the competitors on all datasets, and its overall ratio and recall are also better than those of its competitors. Moreover, we find that either query time, overall ratio, or recall depend only slightly on the dataset dimensionality. For instance, *Audio*, *MNIST*, and *Cifar* have nearly the same cardinalities, but different dimensionality, i.e., 192, 784, and 1024. However, the query times of PM-LSH on them are close. In Table 3, we can see that the dataset *NUS* and *GIST* have large LID values and small RC values, so they are considered as challenging datasets. As shown in Table 4, they have larger query times than the other datasets.

Effect of k. In this set of experiments, we study the performance when varying the value of k in $\{1, 10, 20, \cdots, 100\}$. Due to the space limitation, we only report the performance on three datasets, i.e., *Deep*, *Cifar*, and *Trevi*. The results are shown in Figs. 7-9. In the *Cifar* and *Trevi* datasets, we can see that PM-LSH achieves the best performance on all the aspects. SRS is the second-best algorithm. When using the *Deep* dataset, PM-LSH has the smallest query time and overall ratio, and its recall is close to that of SRS.

As k increases, all algorithms achieve a higher overall ratio and a smaller recall, but the query time is relatively steady. In fact, the algorithms return the best k objects from a candidate set whose size exceeds $\beta n + k$. Therefore, a larger k has little affect on the query time but obviously has an adverse effect on the result quality.

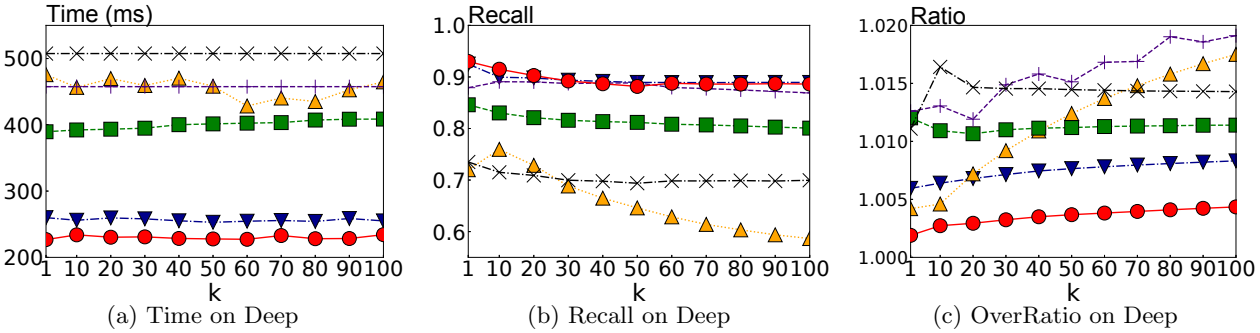
When considered across different datasets with different cardinality n and dimension d, PM-LSH exhibits a consistent high accuracy. This is because PM-LSH is unaffected by the dimensionality of the datasets and because its cost is sublinear in the cardinality of the datasets. In contrast, Multi-Probe is affected significantly by the dimensionality of datasets. The hash number of QALSH is $O(n \log n)$, so its query time increases super-linearly with the dataset cardinality. Similarly, when the dataset cardinality increases, SRS incurs a higher query cost to find an NN in the projected space.

To sum up, PM-LSH has the smallest query time among all competitors. In addition, the accuracy is high. Only SRS is able to achieve a competitive recall in some cases but takes longer query time than PM-LSH.



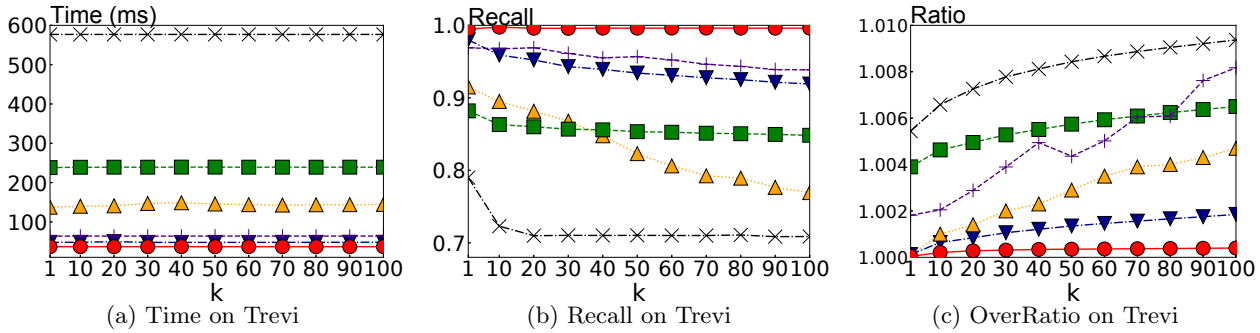
(b) Time on Cifar (c) Recall on Cifar (d) OverRatio on Cifar

Figure 7: Performance on Cifar when Varying k



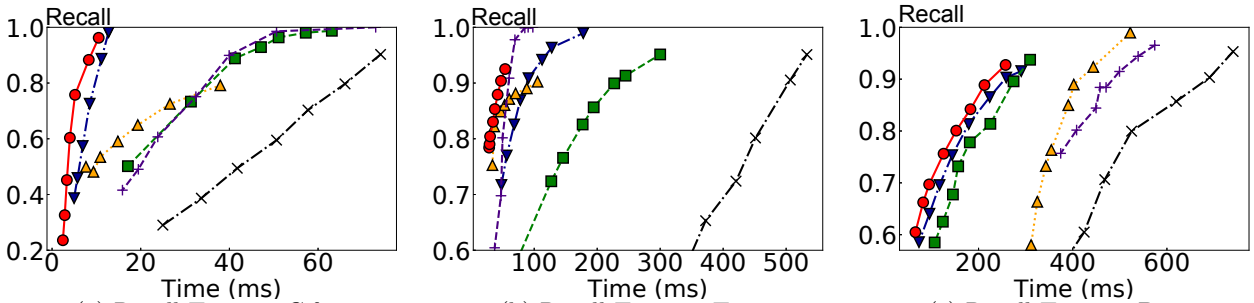
(a) Time on Deep (b) Recall on Deep (c) OverRatio on Deep

Figure 8: Performance on Deep when Varying k



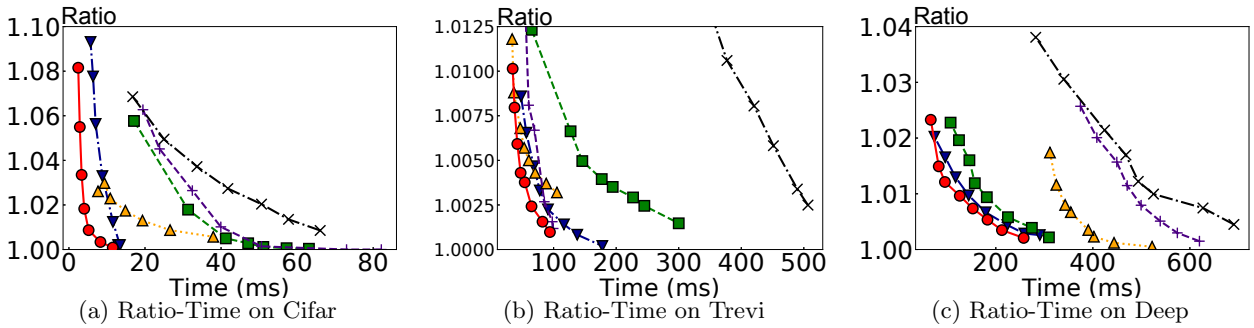
(a) Time on Trevi (b) Recall on Trevi (c) OverRatio on Trevi

Figure 9: Performance on Trevi when Varying k



(a) Recall-Time on Cifar (b) Recall-Time on Trevi (c) Recall-Time on Deep

Figure 10: Recall-Time Curve for Three Datasets



(a) Ratio-Time on Cifar (b) Ratio-Time on Trevi (c) Ratio-Time on Deep

Figure 11: Ratio-Time Curve for Three Datasets

Recall-Time and OverallRatio-Time Curves. In this set of experiments, we evaluate the relationship between the recall or overall ratio and the query time for (c, k)-ANN queries on all the datasets when varying c to obtain different query times. The results are shown in Fig. 10 and Fig. 11. As the tradeoff between the query quality and the query time is the key tradeoff, the LSH methods focus on returning a relatively good result with a much smaller time than those of exact NN algorithms. The results show that all algorithms return more accurate results when more query time is used. They also show that PM-LSH achieves superior efficiency and accuracy when compared to SRS, QALSH, and Multi-Probe. This can be explained as follows. First, PM-LSH has a better distance estimator than QALSH and Multi-Probe, so PM-LSH outperforms them with the same number of retrieved points. Second, PM-LSH needs lower time to obtain the same number of retrieved points since only one or two range queries are required. In contrast, SRS needs T rounds of incremental NN search.

7. RELATED WORK

7.1 Additional LSH Methods

Locality-Sensitive Hashing (LSH) is a prominent approach to speeding up the processing of approximate nearest neighbor querying [4, 9, 10, 12, 22]. LSH was originally proposed by Indyk et al. [18] for the use in Hamming space, and it has since attracted substantial attention due to its excellent performance. Datar et al. [9] propose an LSH function based on p-stable distributions in Euclidean space, which has become a mainstream method that yields low computation cost, a simple geometric interpretation, and a good quality guarantee. Since then, many LSH methods build on this work to choose hash functions [11, 14, 17, 22, 31, 33]. In addition to the competitors introduced in Section 3, other proposals also deserve mention. Based on a rigorous theoretical analysis, Panigrahy et al. [25] propose an entropy-based LSH, and Satuluri et al. [29] propose a BayesLSH. The former tries to reduce the number of hash tables by using multiple perturbed queries, and the latter aims to reduce the query time by estimating the similarity between data and query objects based on Bayes rule. However, both yield limited performance improvements as the assumptions made on the underlying dataset are hard to satisfy and verify. Another interesting proposal is LazyLSH [35], which supports queries in multiple l_p spaces by using one index, thus effectively reducing the space overhead. Another line of hashing-based methods is learning to hash (L2H) [34], which is orthogonal to our work. LSH uses predefined hash functions without considering the underlying dataset, while L2H learns tailored data dependent hash functions. Many learning algorithms have been proposed, such as iterative quantization (ITQ) [13], and generate-to-probe QD ranking (GQR) [20].

7.2 Applications of LSH

Several novel applications of LSH have been put forward in recent years. As processing similarity search on streaming data from Twitter is challenging due to an extremely heavy workload, two studies [26,32] utilize LSH and its variants to support the querying of high throughput streaming data. In addition, a study [1] proposes an LSH scheme that matches two prominent bibliographic databases at paper level by detecting exact matches without false positive. Likewise, a study [27] explores the use of MinHash LSH to index and search Web data. Finally, a study [28] considers an application that identifies potential earthquakes by searching similar time series segments based on the high waveform similarity between reoccurring earthquakes.

8. CONCLUSION

We present a fast and accurate framework, called PM-LSH, for computing (c, k)-ANN queries with theoretical guarantee on the result quality. First, we adopt the PM-tree to index the data points to be queried in a projected space. Second, in order to improve the distance estimation accuracy in the projected space, we develop a tunable confidence interval on the projected distance w.r.t. a given original distance. Finally, we propose an efficient algorithm to search the PM-tree range queries. The experimental study using 7 widely used datasets shows that PM-LSH outperforms five competitors in terms of both query efficiency and result accuracy. Specifically, PM-LSH improves the query time by an average 30% when compared to the closest competitor (SRS). When all the competitors are given the approximately same query time, PM-LSH improves the recall by about 10% when compared to the closest competitor (SRS).

Acknowledgments

This research is supported in part by the NSFC (Grant No. 61902134), the Fundamental Research Funds for the Central Universities (HUST: Grants No. 2019kfyXKJC021, 2019kfyXJJS091). Dr Nguyen Quoc Viet Hung is supported by the ARC DECRA (Grant No. DE200101465).

9. REFERENCES

[1] M. A. Abdulhayoglu and B. Thijs. Use of locality sensitive hashing (LSH) algorithm to match web of science and scopus. *Scientometrics*, 116(2):1229–1245, 2018.

[2] L. Amsaleg, O. Chelly, T. Furon, S. Girard, M. E. Houle, K. Kawarabayashi, and M. Nett. Estimating local intrinsic dimensionality. In *KDD*, pages 29–38, 2015.

[3] A. Andoni and P. Indyk. LSH algorithm and implementation (E2LSH), 2016.

[4] M. Bawa, T. Condie, and P. Ganesan. LSH forest: self-tuning indexes for similarity search. In *WWW*, pages 651–660, 2005.

[5] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, pages 322–331, 1990.

[6] L. Chen, Y. Gao, X. Li, C. S. Jensen, and G. Chen. Efficient metric indexing for similarity search. In *ICDE*, pages 591–602, 2015.

[7] P. Ciaccia, M. Patella, and P. Zezula. A cost model for similarity queries in metric spaces. In *PODS*, pages 59–68, 1998.

[8] A. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. In *WWW*, pages 271–280, 2007.

[9] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Symposium on Computational Geometry*, pages 253–262, 2004.

[10] W. Dong, Z. Wang, W. Josephson, M. Charikar, and K. Li. Modeling LSH for performance tuning. In *CIKM*, pages 669–678, 2008.

[11] J. Gan, J. Feng, Q. Fang, and W. Ng. Locality-sensitive hashing scheme based on dynamic collision counting. In *SIGMOD*, pages 541–552, 2012.

[12] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999.

[13] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *TPAMI*, 35(12):2916–2929, 2013.

[14] P. Haghani, S. Michel, and K. Aberer. Distributed similarity search in high dimensions using locality sensitive hashing. In *EDBT*, pages 744–755, 2009.

[15] J. Harris and H. Stöcker. *Handbook of mathematics and computational science*. Springer, 1998.

[16] J. He, S. Kumar, and S. Chang. On the difficulty of nearest neighbor search. In *ICML*, 2012.

[17] Q. Huang, J. Feng, Y. Zhang, Q. Fang, and W. Ng. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *PVLDB*, 9(1):1–12, 2015.

[18] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998.

[19] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*, pages 2130–2137, 2009.

[20] J. Li, X. Yan, J. Zhang, A. Xu, J. Cheng, J. Liu, K. K. W. Ng, and T. Cheng. A general and efficient querying method for learning to hash. In *SIGMOD*, pages 1333–1347, 2018.

[21] W. Li, Y. Zhang, Y. Sun, W. Wang, M. Li, W. Zhang, and X. Lin. Approximate nearest neighbor search on high dimensional data - experiments, analyses, and improvement. *TKDE*, pages 1–14, 2019.

[22] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe LSH: efficient indexing for high-dimensional similarity search. In *PVLDB*, pages 950–961, 2007.

[23] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Intelligent probing for locality sensitive hashing: Multi-probe LSH and beyond. *PVLDB*, 10(12):2021–2024, 2017.

[24] A. Narang and S. Bhattacherjee. Real-time approximate range motif discovery & data redundancy removal algorithm. In *EDBT*, pages 485–496, 2011.

[25] R. Panigrahy. Entropy based nearest neighbor search in high dimensions. In *SODA*, pages 1186–1195, 2006.

[26] S. Petrovic, M. Osborne, and V. Lavrenko. Streaming first story detection with application to Twitter. In *HLT-NAACL*, pages 181–189, 2010.

[27] B. Rao and E. Zhu. Searching web data using minhash LSH. In *SIGMOD*, pages 2257–2258, 2016.

[28] K. Rong, C. E. Yoon, K. J. Bergen, H. Elezabi, P. Bailis, P. Levis, and G. C. Beroza. Locality-sensitive hashing for earthquake detection: A case study scaling data-driven science. *PVLDB*, 11(11):1674–1687, 2018.

[29] V. Satuluri and S. Parthasarathy. Bayesian locality sensitive hashing for fast similarity search. *PVLDB*, 5(5):430–441, 2012.

[30] T. Skopal, J. Pokorný, and V. Snásel. Nearest neighbours search using the PM-tree. In *DASFAA*, pages 803–815, 2005.

[31] Y. Sun, W. Wang, J. Qin, Y. Zhang, and X. Lin. SRS: solving c-approximate nearest neighbor queries in high dimensional euclidean space with a tiny index. *PVLDB*, 8(1):1–12, 2014.

[32] N. Sundaram, A. Turmukhametova, N. Satish, T. Mostak, P. Indyk, S. Madden, and P. Dubey. Streaming similarity search over one billion tweets using parallel locality-sensitive hashing. *PVLDB*, 6(14):1930–1941, 2013.

[33] Y. Tao, K. Yi, C. Sheng, and P. Kalnis. Quality and efficiency in high dimensional nearest neighbor search. In *SIGMOD*, pages 563–576, 2009.

[34] J. Wang, T. Zhang, J. Song, N. Sebe, and H. T. Shen. A survey on learning to hash. *TPAMI*, 40(4):769–790, 2018.

[35] Y. Zheng, Q. Guo, A. K. H. Tung, and S. Wu. Lazylsh: Approximate nearest neighbor search for multiple distance functions with a single index. In *SIGMOD*, pages 2023–2037, 2016.