LazyLSH: Approximate Nearest Neighbor Search for Multiple Distance Functions with a Single Index

Yuxin Zheng[†], Qi Guo[†], Anthony K. H. Tung[†] and Sai Wu[#]
[†] School of Computing, National University of Singapore, Singapore
[#] College of Computer Science and Technology, Zhejiang University, China
[†]{yuxin, qiguo, atung}@comp.nus.edu.sg, [#]wusai@zju.edu.cn

ABSTRACT

Due to the "curse of dimensionality" problem, it is very expensive to process the nearest neighbor (NN) query in high-dimensional spaces; and hence, approximate approaches, such as Locality-Sensitive Hashing (LSH), are widely used for their theoretical guarantees and empirical performance. Current LSH-based approaches target at the ℓ_1 and ℓ_2 spaces, while as shown in previous work, the fractional distance metrics (ℓ_p metrics with $0 < p < 1$) can provide more insightful results than the usual ℓ_1 and ℓ_2 metrics for data mining and multimedia applications. However, none of the existing work can support multiple fractional distance metrics using one index. In this paper, we propose LazyLSH that answers approximate nearest neighbor queries for multiple ℓ_p metrics with theoretical guarantees. Different from previous LSH approaches which need to build one dedicated index for every query space, LazyLSH uses a single base index to support the computations in multiple ℓ_p spaces, significantly reducing the maintenance overhead. Extensive experiments show that LazyLSH provides more accurate results for approximate kNN search under fractional distance metrics.

CCS Concepts

•Information systems → Nearest-neighbor search;

Keywords

Locality sensitive hashing, Nearest neighbor search, ℓ_p metrics

1. INTRODUCTION

State-of-the-art kNN processing techniques have been proposed for low-dimensional cases. However, due to the "curse of dimensionality", the same techniques cannot be directly applied to high-dimensional spaces. It was shown that conventional kNN processing approaches become even slower than the naive linear-scan approach [18]. One compromise solution is to adopt the approximate kNN technique which returns k points within distance cR from a query point, where c is an approximation ratio and R is the distance between the query point and its true (k)th nearest neighbor. The intuition is that in high-dimensional spaces, approximate results are good enough for most applications.

Table 1: Classification accuracy

Dataset	Classification accuracy (%)						
	Real 1NN	LazyLSH (Approximate 1NN)					
	$\ell_{1.0}$	$\ell_{0.5}$	$\ell_{0.6}$	$\ell_{0.7}$	$\ell_{0.8}$	$\ell_{0.9}$	$\ell_{1.0}$
Ionos	90.9	**92.0**	91.7	91.7	91.7	91.7	91.5
Musk	93.5	**94.0**	93.8	93.5	93.4	93.4	93.5
BCW	92.8	**93.3**	**93.3**	93.1	93.0	92.6	92.8
SVS	67.5	67.8	**68.9**	67.8	67.4	67.2	67.5
Segme	91.9	92.1	92.1	**92.4**	92.3	92.1	91.9
Giset	96.2	94.9	95.7	96.4	96.4	**96.8**	96.5
SLS	90.0	87.8	88.3	88.7	89.2	**90.0**	89.8
Sun	9.5	9.0	9.3	9.3	9.4	9.4	**9.5**
Mnist	96.3	95.1	95.4	95.7	95.9	96.0	**96.2**

To process approximate kNN queries, several methods have been proposed [5, 26, 2, 33], among which, locality-sensitive hashing (LSH) [26] is widely used for its theoretical guarantees and empirical performance. In essence, the LSH scheme is based on a set of hash functions from the *locality-sensitive hash family* which guarantees that similar points are hashed into the same buckets with higher probabilities than dissimilar points. The LSH scheme was first proposed by Indyk et al. [26] for the use in the binary Hamming space, and later was extended for the use in the Euclidean space by Datar et al. [18] based on the p-stable distribution.

It was observed that the effectiveness of high-dimensional search is sensitive to the choice of distance functions [1]. Although the Manhattan (ℓ_1) and Euclidean (ℓ_2) metrics are widely used, it was shown that ℓ_p metrics with $0 < p < 1$, called *fractional distance metrics*, can provide more insightful results from both theoretical and empirical perspectives for data mining applications [1, 16] and content-based image retrievals [25]. Furthermore, it was shown that the optimal ℓ_p metric is application-dependent and required to be tuned or adjusted for each application [1, 16, 25, 20].

As an example, Table 1 shows the accuracy of the kNN classifier [17] under different ℓ_p metrics. We test *Mnist* [29], *Sun* [19] and seven datasets from the UCI ML repository[1]. The ground-truth classification results are provided by the

SIGMOD '16, June 26–July 1, 2016, San Francisco, CA, USA.
© 2016 ACM. ISBN 978-1-4503-3531-7/16/06...$15.00
DOI: http://dx.doi.org/XXXX.XXXX

[1] http://archive.ics.uci.edu/ml/
The used datasets are: Ionosphere (Ionos), Musk, Breast Cancer Wisconsin (BCW), Statlog Vehicle Silhouettes (SVS), Segmentation (Segme), Gisette (Giset) and Statlog Landsat Satellite (SLS).

datasets themselves. For each query point, we retrieve its nearest neighbor and assign it to the same class tag as its nearest neighbor. For ℓ_p metrics ($0.5 \leq p \leq 1$), we compute the approximate 1NN using our LazyLSH technique proposed in the paper. For comparison, we also show the results of the 1NN classifiers where the 1NN is the true 1NN in the ℓ_1 space. We highlight the highest accuracy for LazyLSH in bold font. The results indicate that the best classification result may be obtained using different fractional distance metrics for different datasets. There is no way to know which fractional distance is optimal for a specific dataset. This finding is similar to the observations presented in [1, 25]. Therefore, before implementing a system, we need an approach that can explore the data using different distance metrics, such that we can select a proper one to achieve the best mining results.

Unfortunately, due to the lack of closed form density functions for p-stable distributions when $p \neq 1$ or 2, it is nontrivial to generate p-stable random variables and build an optimal index structure for fractional distance metrics. Moreover, the conventional approach of building one index for each possible value of p will incur very high costs in terms of computational time and space requirement (with the number of possible values of p being potentially infinite). To address this problem, in this paper, we propose LazyLSH to process approximate kNN queries in different ℓ_p spaces using only one single index.

LazyLSH builds an LSH index in a predefined ℓ_{p_0} space, which is referred to as the base space. Using this materialized index, LazyLSH can answer approximate kNN queries in a user-specific query space. The word "Lazy" is borrowed from the lazy learning algorithms [49] in which generalization beyond the training data is delayed until a query is issued. LazyLSH means that we do not build an index for every query space. Instead, we reuse the index constructed in the base space to answer queries in the query space. Our analysis shows that if two points are close in an ℓ_{p_1} space, then they are likely to be close in another ℓ_{p_2} space. We also find that a locality-sensitive hash function built in the base space is still locality-sensitive in the query space when certain conditions hold. With this observation, LazyLSH adopts the strategy of having "**one index for many fractional distance metrics**". Figure 1 illustrates this idea. A single materialized LSH index is built using a specific distance function, based on which, we can approximately process kNN queries for other fractional distance metrics.

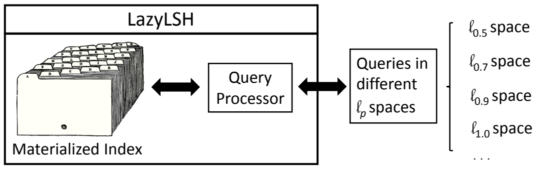


Figure 1: LazyLSH Overview

In order to get more precise results, we propose a method called query-centric rehashing to search the base index and retrieve nearby objects. We further observe that during the processing of queries under different ℓ_p metrics, many common index entries are probed. This finding motivates us to optimize the processing of multiple queries under different ℓ_p metrics concurrently by sharing their I/Os.

We summarize the contributions of this paper as follows.

- We propose a novel method called LazyLSH to answer approximate nearest neighbor queries under multiple ℓ_p metrics. Compared to the costly naive method which builds an LSH index for every value of p to cover all possible fractional distance metrics, LazyLSH maintains only a single copy of LSH index in the base space, significantly reducing the storage overhead.

- We give a theoretical proof that when certain conditions hold, locality-sensitive hash function can be extended to support the fractional distance metrics. This is the first work that gives a theoretical bound for the approximate kNN processing using LSH with the fractional distance metric.

- We propose two novel optimization methods, namely query-centric rehashing and multi-query optimization, to improve the effectiveness and efficiency of performing queries.

- We experimentally verify the effectiveness and efficiency of our proposed LazyLSH using both synthetic and real datasets. Experimental results show that LazyLSH provides more accurate results for approximate kNN search under fractional distance metrics, and it can be used as the supervision to optimally choose the ℓ_p metric for different applications.

The rest of this paper is organized as follows. Section 2 briefly reviews the preliminaries on LSH. Section 3 presents the technical details of the proposed LazyLSH method. Section 4 shows the processing of approximate range queries and approximate kNN queries. Then, we experimentally evaluate the proposed method in Section 5 and discuss the related studies in Section 6. Finally, we conclude this paper in Section 7. For the ease of presentation, we summarize our notations in Table 2.

2. PRELIMINARY

Before delving into the details of LazyLSH, we first review some preliminary knowledge of the locality-sensitive hashing (LSH) method. We begin with the definition of the ℓ_p distance used in this paper.

Definition 1 (ℓ_p distance). *The distance between any two d-dimensional points $\vec{o}$ and $\vec{q}$ in the ℓ_p space, denoted as $\ell_p(\vec{o}, \vec{q})$, is computed as:*

$$\ell_p(\vec{o}, \vec{q}) = \sqrt[p]{\sum_{i=1}^{d} |o_i - q_i|^p} \tag{1}$$

If $0 < p < 1$, $\ell_p(\vec{o}, \vec{q})$ is called the *fractional distance metric* [1]. In similarity search, if $\ell_p(\vec{o}, \vec{q}) \leq r$, we say the point $\vec{o}$ is within the *ball* of radius r centered at the point $\vec{q}$, denoted as $B_p(\vec{q}, r)$.

Definition 2 (Ball $B_p(\vec{q}, r)$). *Given a point $\vec{q} \in \mathbb{R}^d$, and a radius r, the ball of radius r centered at point $\vec{q}$ in the ℓ_p space is defined as $B_p(\vec{q}, r) = \{\vec{v} \in \mathbb{R}^d | \ell_p(\vec{v}, \vec{q}) \leq r\}$.*

LSH methods try to map the points within a ball to the same hash bucket. Let $\mathcal{H}$ be a family of functions mapping $\mathbb{R}^d$ to some universe U. For any two points $\vec{o}, \vec{q} \in \mathbb{R}^d$, consider a process in which we choose a function h from $\mathcal{H}$ at

Table 2: Table of Notations

$\mathcal{D}$	database
d	dimensionality
$\vec{q}$	query point
$h_i^*(\cdot)$	based materialized hash function
$\vec{a}$	random vector in the hash function
b	offset in the hash function
c	approximation ratio
X	random variable
$\ell_p(\vec{o}, \vec{q})$	the ℓ_p distance between $\vec{o}$ and $\vec{q}$
p	the subscript used in the ℓ_p space or ℓ_p distance
δ	radius in the ℓ_p space
r	radius in the ℓ_1 space
$\delta^{\perp}$	lower bound of the ℓ_1 distance given $\ell_p = \delta$
$\delta^{\top}$	upper bound of the ℓ_1 distance given $\ell_p = \delta$
$B_p(\vec{q}, r)$	the ball of radius r centered at $\vec{q}$ in the ℓ_p space
η	the number of required hash functions
θ	the collision count threshold

random, and analyze the probability of $h(\vec{o}) = h(\vec{q})$. The family $\mathcal{H}$ is called locality-sensitive if it satisfies the following conditions.

Definition 3 (Locality-sensitive hashing). *Let $d(\cdot,\cdot)$ be a distance function of a metric space. A family $\mathcal{H}$ is called (r, cr, p_1, p_2)-sensitive if for any two points $\vec{o}, \vec{q} \in \mathbb{R}^d$, satisfying*
(1) if $d(\vec{o}, \vec{q}) \leq r$, then $Pr_{\mathcal{H}}[h(\vec{o}) = h(\vec{q})] \geq p_1$,
(2) if $d(\vec{o}, \vec{q}) > cr$, then $Pr_{\mathcal{H}}[h(\vec{o}) = h(\vec{q})] < p_2$,
(3) $c > 1$, and
(4) $p_1 > p_2$.

Various LSH families have been discovered for different distance metrics [4]. In particular, the LSH family for the ℓ_p distance is found based on the p-stable distribution [18].

Definition 4 (p-stable distribution). *A distribution G over $\mathbb{R}$ is called* p-stable, *if there exists $p \geq 0$ such that for any n real numbers $v_1, ..., v_n$ and i.i.d. random variables $X_1, ..., X_n$ with distribution G, the variable $\sum_i v_i X_i$ has the same distribution as the variable $(\sum_i |v_i|^p)^{1/p} X$, where X is a random variable with distribution G. It has been proved that stable distributions exist for $p \in (0, 2]$. In particular,*
- *The Cauchy distribution, with a density function $f(x) = \frac{1}{\pi}\frac{1}{1+x^2}$, is 1-stable;*
- *The Gaussian distribution, with a density function $f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$, is 2-stable.*

Using the p-stable distribution, one can generate a d-dimensional vector $\vec{a}$ by setting its element as a random value from the p-stable distribution. Given two points $\vec{v_1}, \vec{v_2} \in \mathbb{R}^d$, $(\vec{a}.\vec{v_1} - \vec{a}.\vec{v_2})$ is distributed as $\ell_p(\vec{v_1}, \vec{v_2})X$, where X is a random variable with the same p-stable distribution. Based on the above observations, Datar et al. [18] proposed the following LSH family for the ℓ_p distance. A data point in $\mathbb{R}^d$ is projected onto a random line $\vec{a}$, which is segmented into equi-width intervals with length r_0. Formally, the proposed LSH function in the ℓ_p space is defined as:

$$h(\vec{v}) = \lfloor \frac{\vec{a}.\vec{v} + b}{r_0} \rfloor, \qquad (2)$$

where the projection vector $\vec{a} \in \mathbb{R}^d$ is constructed by picking each coordinate from a p-stable distribution.

Given $\vec{v_1}, \vec{v_2} \in \mathbb{R}^d$, let $s = \ell_p(\vec{v_1}, \vec{v_2})$. The probability that $\vec{v_1}$ and $\vec{v_2}$ collide under a hash function $h(\cdot)$, denoted as $p(s, r_0)$, can be computed as follows:

$$p(s, r_0) = \int_0^{r_0} \frac{1}{s} f_p(\frac{t}{s})(1 - \frac{t}{r_0}) dt, \qquad (3)$$

where $f_p(\cdot)$ is the probability density function of the absolute value of the p-stable distribution, and $p(s, r_0)$ is monotonically decreasing with s when r_0 is fixed [18]. As a result, the LSH family of the ℓ_p distance is $(1, c, p_1, p_2)$-sensitive with $p_1 = p(1, r_0)$ and $p_2 = p(c, r_0)$. For special cases such as p equals to 1 and 2, we can compute the probabilities using the corresponding density functions [18].

- For the Cauchy distribution ($p = 1$), we have

$$p(s, r_0) = 2\frac{\arctan(r_0/s)}{\pi} - \frac{1}{\pi(r_0/s)} \ln(1 + (r_0/s)^2) \qquad (4)$$

- For the Gaussian distribution ($p = 2$), we have

$$p(s, r_0) = 1 - 2norm(-r_0/s) - \frac{2}{\sqrt{2\pi}(r_0/s)}(1 - e^{-(r^2/2s^2)}), \qquad (5)$$

where $norm(\cdot)$ is the cumulative distribution function of the standard normal distribution.

The above LSH family was employed to process approximate nearest neighbor queries which are defined as:

Definition 5 ($\mathcal{N}_p(\vec{q}, k, c)$ problem). *Given a dataset $\mathcal{D}$, a point $\vec{q} \in \mathbb{R}^d$, a cardinality k, an approximate ratio c, and an ℓ_p space, the c-approximate k nearest neighbors search returns a set of k points, $\mathcal{N}_p(\vec{q}, k, c) = \{\vec{o_1}, \ldots, \vec{o_k}\}$, where points are sorted in ascending order of their distances to $\vec{q}$ in the ℓ_p space, and $\vec{o_i}$ is a c-approximation of the real i^{th} nearest neighbor. Let $\vec{o_1^*}, \ldots, \vec{o_k^*}$ be the real kNNs in ascending order of their distances to $\vec{q}$. Then $\ell_p(\vec{o_i}, \vec{q}) \leq (c \times \ell_p(\vec{o_i^*}, \vec{q}))$ holds for all $i \in [1, k]$.*

Several approaches were proposed to answer approximate nearest neighbor queries in the Euclidean space [18, 32, 45, 21, 43]. We briefly discuss two related methods: E2LSH [18] and C2LSH [21].

E2LSH: E2LSH [18] is the first proposed LSH method to answer the $\mathcal{R}_p(\vec{q},\ r,\ c)$ problem in the Euclidean space where $p = 2$, which is defined as:

Definition 6 ($\mathcal{R}_p(\vec{q},\ r,\ c)$ problem). *Given a dataset $\mathcal{D}$, a query point $\vec{q} \in \mathbb{R}^d$, $\mathcal{R}_p(\vec{q},\ r,\ c)$ returns a point $\vec{o'} \in \mathcal{D}$, where $\vec{o'} \in B_p(\vec{q}, cr)$, if there exists a point $\vec{o} \in B_p(\vec{q}, r)$.*

As can be seen, the $\mathcal{R}_p(\vec{q},\ r,\ c)$ problem is a decision version of the $\mathcal{N}_p(\vec{q}, k, c)$ problem. E2LSH exploits LSH functions to address the $\mathcal{R}_2(\vec{q},\ r,\ c)$ problem in the following way. First, a set of m LSH functions $h_1(\cdot), \ldots, h_m(\cdot)$ are randomly chosen from an (r, cr, p_1, p_2)-sensitive family $\mathcal{H}$, and they are concatenated to form a compound hash function $g(\cdot)$, where $g(\vec{p}) = (h_1(\vec{p}), \ldots, h_m(\vec{p}))$ for a point $\vec{p} \in \mathbb{R}^d$. By using a compound hash function instead of a single LSH function, the probability that two faraway points collide can be largely reduced. Then, the hash function $g(\cdot)$ is used to map all the data to a hash table. The above two steps are repeated for L times and accordingly, L compound hash functions $g_1(\cdot), \ldots, g_L(\cdot)$ are used to produce L hash tables.

When a query $\vec{q}$ comes, the points from buckets $g_1(\vec{q}), \ldots, g_L(\vec{q})$ are retrieved until all the points or the first $3L$ points are found. For each retrieved point $\vec{v}$, it is returned if $\vec{v} \in B_2(\vec{q}, cr)$. An $\mathcal{N}_2(\vec{q}, k, c)$ query can be answered by issuing a series of $\mathcal{R}_2(\vec{q},\ c,\ r)$ queries using gradually increasing search radii. For this purpose, hash tables with different radii must be built, incurring a high storage cost.

C2LSH: To avoid building many hash tables for different radii, C2LSH [21] is proposed by changing the original hash function to:

$$h(\vec{v}) = \lfloor \frac{\vec{a}.\vec{v} + b^*}{r_0} \rfloor, \tag{6}$$

where b^* is uniformly drawn from $\lfloor 0, c^{\lceil \log_c td \rceil}(r_0)^2 \rfloor$, c is the approximation ratio, t is the largest coordinate value, and d is the dimensionality of the data. It is proved that the hash function is $(1,\ c,\ p_1,\ p_2)$-sensitive.

First, a materialized index is built for a set of base LSH functions with a small interval r_0. Then, C2LSH reuses the materialized index to retrieve objects at different radii without explicitly building hash tables for different radii. This process is referred to as *virtual rehashing.* To answer an $\mathcal{R}_2(\vec{q},\ r,\ c)$ query, C2LSH modifies the hash function as:

$$H^r(\vec{v}) = \lfloor \frac{h(\vec{v})}{r} \rfloor \tag{7}$$

Note that the $H^r(\vec{v})$ is $(r,\ cr,\ p_1,\ p_2)$-sensitive. *Virtual rehashing* simplifies the process of retrieving the objects hashed to $H^r(\vec{q})$ by guaranteeing that it is identical to retrieving objects in the buckets within $[\lfloor \frac{h_{\vec{a},b}(\vec{v})}{r} \rfloor \times r, \lfloor \frac{h_{\vec{a},b}(\vec{v})}{r} \rfloor \times r + r - 1]$ in the base hash function.

In addition, C2LSH estimates the probability of being the nearest neighbor using the *collision count.* If the number of an object colliding with a query exceeds a certain threshold, namely the *collision count threshold* θ, the object is likely to be a neighbor. Such an object is considered as a candidate and retrieved for computing its real distance to the query.

As a result, an $\mathcal{N}_2(\vec{q}, k, c)$ query can be answered by C2LSH by issuing a set of $\mathcal{R}_2(\vec{q},\ c,\ r)$ queries with increasing radii. C2LSH is claimed to be correct if these two properties hold with a constant probability.

- $\mathcal{P}_1$: If $\vec{v} \in B_2(\vec{q}, r)$, then the number of $\vec{v}$'s collision with the query $\vec{q}$ is at least θ.
- $\mathcal{P}_2$: The total number of false positives is smaller than $\beta|\mathcal{D}|$, where $|\mathcal{D}|$ is the cardinality of the database $\mathcal{D}$.

In $\mathcal{P}_1$, the number of collisions θ is related to the number of base hash functions, which is denoted as η. Given an error probability ε and a false positive rate β, θ and η must be carefully tuned for the best performance.

LEMMA 1. *If η and θ are set to:*

$$\eta = \lceil \frac{\ln \frac{1}{\varepsilon}}{2(p_1 - p_2)^2}(1+z)^2 \rceil, \text{ where } z = \sqrt{\frac{\ln \frac{2}{\beta}}{\ln \frac{1}{\varepsilon}}}, \tag{8}$$

$$\theta = \frac{zp_1 + p_2}{1+z}\eta, \tag{9}$$

then $Pr[\mathcal{P}_1] \geq 1 - \varepsilon$ and $Pr[\mathcal{P}_2] \geq 0.5$. [21]

Therefore, both $\mathcal{P}_1$ and $\mathcal{P}_2$ hold with a constant probability when the parameters are set as above, and C2LSH can correctly answer the $\mathcal{N}_2(\vec{q},\ c,\ r)$ query.

3. LAZYLSH

In this section, we present LazyLSH as an efficient mechanism to process approximate kNN queries in different ℓ_p spaces. We begin with an overview, and then illustrate the technical details of LazyLSH.

3.1 Overview

In previous work such as E2LSH and C2LSH, an LSH index is built for the ℓ_2 space, while Aggarwal et al. [1] showed that the fractional distance metrics $(0 < p < 1)$ provide more meaningful results and improve the effectiveness of information retrieval algorithms. Extending techniques in E2LSH and C2LSH to support an arbitrary fractional distance metric is not a trivial task. In this paper, we present LazyLSH, a novel approach that can process approximate nearest neighbor queries using different fractional distance metrics with a single LSH index.

LazyLSH is proposed based on the intuition that given $p, s > 0$, if two points are close in the ℓ_p space, then they are likely to be close in the ℓ_s space. Let $\epsilon = |p - s|$. The property holds with a higher probability for a smaller ϵ. This property can be extended as: the $(r,\ cr,\ p_1,\ p_2)$-sensitive hash function in the ℓ_p space is $(\delta,\ c\delta,\ p_1',\ p_2')$-sensitive in the ℓ_s space if certain conditions hold. We will give a detailed theoretical analysis for this property later in this section.

Using this property, LazyLSH can transform the hash functions between different ℓ_p spaces. LSH families for the ℓ_1 and ℓ_2 metrics have been well studied [4, 45, 21]. As the ℓ_1 metric is closer to the fractional distance metrics, in LazyLSH, we materialize the LSH index in the ℓ_1 space as our base index. We generate η_p base hash functions $h_1^*(\cdot)$, $\ldots$, $h_{\eta_p}^*(\cdot)$, where the setting of η_p will be discussed later. In particular, we construct $h_i^*(\cdot)$ as:

$$h_i^*(\vec{v}) = \lfloor \frac{\vec{a_i}.\vec{v} + b_i^*}{r_0} \rfloor, \tag{10}$$

where a_i is a random vector whose each entry is drawn from the 1-stable (Cauchy) distribution and the other parameters are the same as the ones in Equation 6. Each base hash function h_i^* is $(1,\ c,\ p_1,\ p_2)$-sensitive in the ℓ_1 space, with $p_1 = p(1, r_0)$, $p_2 = p(c, r_0)$ as presented in Equation 4.

Using the materialized index, LazyLSH can answer $\mathcal{N}_p(\vec{q}, k, c)$ queries with probabilistic guarantees. For the ease of presentation, we first show our observation that a $(r,\ cr,\ p_1,\ p_2)$-sensitive hash function in the ℓ_p space is $(\delta,\ c\delta,\ p_1',\ p_2')$-sensitive in another ℓ_s space in this section. Then we illustrate how the LazyLSH method answers $\mathcal{R}_p(\vec{q}, r, c)$ and $\mathcal{N}_p(\vec{q}, k, c)$ queries in Section 4.

3.2 LSH in an ℓ_p Space

If there exists a point $\vec{o} \in B_p(\vec{q}, \delta)$, an $\mathcal{R}_p(\vec{q},\ \delta,\ c)$ query returns a point $\vec{o'}$ if $\vec{o'} \in B_p(\vec{q}, c\delta)$. We observe that $B_p(\vec{q}, \delta)$ and $B_1(\vec{q}, r)$ share a lot of common areas if r is carefully tuned for δ. Figure 2 plots a $B_1(\vec{q}, r)$ ball in blue and a $B_p(\vec{q}, \delta)$ ball in red, where $0 < p < 1$. The shadow region represents the intersection of $B_1(\vec{q}, r)$ and $B_p(\vec{q}, \delta)$ which takes over a large portion of $B_p(\vec{q}, \delta)$. This observation motivates us to use a ball $B_1(\vec{q}, r)$ in the ℓ_1 space to approximate $B_p(\vec{q}, \delta)$ in the ℓ_p space for the query $\mathcal{R}_p(\vec{q},\ \delta,\ c)$.

Given two points $\vec{q}$, $\vec{o} \in \mathbb{R}^d$, with $\ell_p(\vec{q}, \vec{o}) = \delta$. We can compute the lower bound and upper bound of δ in the ℓ_1 space, denoted as $\delta^\perp$ and $\delta^\top$ respectively. Figure 3 shows the geometric interpretations of $\delta^\perp$ and $\delta^\top$ for $0 < p < 1$ and $p > 1$. The values of $\delta^\perp$ and $\delta^\top$ are computed as:

$$\delta^\perp = \begin{cases} \frac{d \times \delta}{\sqrt[p]{d}} & \text{if } 0 < p < 1 \\ \delta & \text{if } p \geq 1 \end{cases} \qquad \delta^\top = \begin{cases} \delta & \text{if } 0 < p < 1 \\ \frac{d \times \delta}{\sqrt[p]{d}} & \text{if } p \geq 1 \end{cases} \tag{11}$$

Our goal is to use an ℓ_1 ball $B_1(\vec{q}, r)$ to approximate the

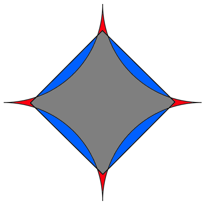
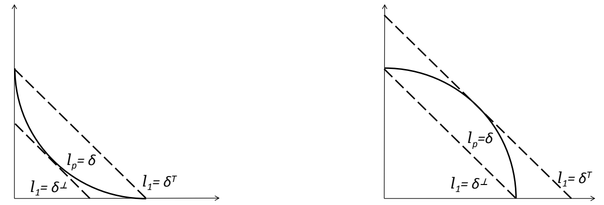


Figure 2: Use $B_1(\vec{q}, r)$ to approximate $B_p(\vec{q}, \delta)$ (Best viewed in color)



(a) $0 < p < 1$

(b) $p > 1$

Figure 3: Bounds of ℓ_p distance

ℓ_p ball $B_p(\vec{q}, \delta)$ specified by the query $\mathcal{R}_p(\vec{q},\ \delta,\ c)$. The radius r significantly affects the search performance. If $r < \delta^{\perp}$, we may fail to retrieve many candidate results, leading to many false negatives. On the other hand, if $r > \delta^{\top}$, many irrelevant objects are retrieved, generating many false positives. Therefore, a proper r should be chosen in the range of $[\delta^{\perp}, \delta^{\top}]$ for $B_1(\vec{q}, r)$ to approximate $B_p(\vec{q}, \delta)$.

Given a query $\mathcal{R}_p(\vec{q},\ \delta, c)$ and a based $(1, c, p_1, p_2)$-sensitive hash function $h_i^*(\cdot)$, we modify $h_i^*(\cdot)$ as follows:

$$h_i^r(\vec{v}) = \lfloor \frac{\vec{a_i}.\vec{v} + b_i^*}{r_0 r} \rfloor, \tag{12}$$

where r is in $[\delta^{\perp}, \delta^{\top}]$ as discussed above. It is easy to see that $h_i^r(\cdot)$ is $(r,\ cr,\ p(r, r_0 r),\ p(cr, r_0 r))$-sensitive in the ℓ_1 space, where $p(\cdot, \cdot)$ is defined in Equation 3. We further observe the following lemma.

Lemma 2. *Let $p(s, r) = \int_0^r \frac{1}{s} f_p(\frac{t}{s})(1 - \frac{t}{r})dt$ as shown in Equation 3. For any real number $c > 0$, $p(s, r) = p(cs, cr)$.*

Proof. See Appendix A.1. □

By Lemma 2, we get $p(r, r_0 r) = p(1, r_0) = p_1$ and $p(cr, r_0 r) = p(c, r_0) = p_2$, where p_1 and p_2 are the same as the ones defined in the base materialized hash function $h_i^*(\cdot)$.

Recall that our LSH index is built in the ℓ_1 space. The LSH family guarantees that close points in the ℓ_1 space are likely to be hashed into the same bucket. We are interested in whether an LSH function in the ℓ_1 space is still locality-sensitive in another ℓ_p space. Given a $(r,\ cr,\ p_1,\ p_2)$-sensitive hash function $h_i^r(\cdot)$ in the ℓ_1 space, we define the following events for any two points $\vec{o}, \vec{q} \in \mathbb{R}^d$:

$e_1 : h_i^r(\vec{o}) = h_i^r(\vec{q})$.
$e_2 : \ell_p(\vec{o}, \vec{q}) \leq \delta$. $\quad e_3 : \ell_p(\vec{o}, \vec{q}) > c\delta$, $\quad$ where $c > 1$
$e_4 : \ell_1(\vec{o}, \vec{q}) \leq r$. $\quad e_5 : \ell_1(\vec{o}, \vec{q}) > cr$, $\quad$ where $c > 1$

To verify whether the modified hash function $h_i^r(\cdot)$ is locality-sensitive in the ℓ_p space, we need to calculate the probability of e_1 given e_2 and the probability of e_1 given e_3, i.e. $Pr(e_1|e_2)$ and $Pr(e_1|e_3)$ respectively. Let e^c represent the complementary of event e. By Bayes' Theorem, we compute a lower bound of $Pr(e_1|e_2)$:

$$\begin{aligned} Pr(e_1|e_2) &= Pr(e_1 \wedge e_4|e_2) + Pr(e_1 \wedge e_4^c|e_2) \\ &= Pr(e_4|e_2)Pr(e_1|e_4 \wedge e_2) + Pr(e_4^c|e_2)Pr(e_1|e_4^c \wedge e_2) \\ &\text{(Note: } Pr(e_1|e_4 \wedge e_2) \geq Pr(e_1|e_4) \geq p_1, \text{ and} \\ &\ell_p(\vec{o}, \vec{q}) \leq \delta \text{ implies } \ell_1(\vec{o}, \vec{q}) \leq \delta^{\top}\ ) \\ &\geq Pr(e_4|e_2)p_1 + (1 - Pr(e_4|e_2))p(\delta^{\top}, r_0 r) \\ &= Pr(e_4|e_2)p_1 + (1 - Pr(e_4|e_2))p(1, \frac{r_0 r}{\delta^{\top}}) \end{aligned} \tag{13}$$

This is because $p(s, r)$ is monotonically decreasing with s when r is fixed. We infer $p(1, \frac{r_0 r}{\delta^{\top}}) \leq p_1$ from $\delta^{\perp} \leq r \leq \delta^{\top}$. For simplicity, we use p_1' to denote this lower bound. Consequently, we get $p_1' \leq p_1$. Similarly, we compute an upper bound of $Pr(e_1|e_3)$:

$$\begin{aligned} Pr(e_1|e_3) =& Pr(e_1 \wedge e_5|e_3) + Pr(e_1 \wedge e_5^c|e_3) \\ =& Pr(e_5|e_3)Pr(e_1|e_5 \wedge e_3) + Pr(e_5^c|e_3)Pr(e_1|e_5^c \wedge e_3) \\ &\text{(Note: } Pr(e_1|e_5 \wedge e_3) \leq Pr(e_1|e_5) \leq p_2, \text{ and} \\ &\ell_p(\vec{o}, \vec{q}) > c\delta \text{ implies } \ell_1(\vec{o}, \vec{q}) > c\delta^{\perp}\ ) \\ \leq& Pr(e_5|e_3)p_2 + (1 - Pr(e_5|e_3))p(c\delta^{\perp}, r_0 r) \\ =& Pr(e_5|e_3)p_2 + (1 - Pr(e_5|e_3))p(c, r_0 r/\delta^{\perp}) \\ \leq& \begin{cases} p_2 & \text{if } p_2 \geq p(c, r_0 r/\delta^{\perp}), \text{ i.e. } r \leq \delta^{\perp} \\ p(c, r_0 r/\delta^{\perp}) & \text{otherwise} \end{cases} \\ &\text{(Note: we set } \delta^{\perp} \leq r \leq \delta^{\top}\ ) \\ =& p(c, r_0 r/\delta^{\perp}) \end{aligned} \tag{14}$$

Then, we use p_2' to denote $p(c, r_0 r/\delta^{\perp})$. Based on the above two inequalities, we have the following theorem:

Theorem 1. *Given an $(r,\ cr,\ p_1,\ p_2)$-sensitive hash function $h_i^r(\cdot)$ in the ℓ_1 space, we find the following two conditions hold for a distance threshold δ in another ℓ_p space, where $\delta^{\perp} \leq r \leq \delta^{\top}$:*
(1) if $\ell_p(\vec{o}, \vec{q}) \leq \delta$, then $Pr[h_i^r(\vec{o}) = h_i^r(\vec{q})] \geq p_1'$,
(2) if $\ell_p(\vec{o}, \vec{q}) > c\delta$, then $Pr[h_i^r(\vec{o}) = h_i^r(\vec{q})] < p_2'$,
where p_1' and p_2' are stated as stated above.

$$\begin{aligned} p_1' &= Pr(e_4|e_2)p_1 + (1 - Pr(e_4|e_2))p(1, \frac{r_0 r}{\delta^{\top}}) \\ p_2' &= p(c, r_0 r/\delta^{\perp}) \end{aligned}$$

3.3 Computing Internal Parameters

To ensure the correctness of LazyLSH, two parameters are required to be computed. One is the radius r of an ℓ_1 ball $B_1(\vec{q}, r)$ to approximate the ℓ_p ball $B_p(\vec{q}, \delta)$ for query $\mathcal{R}_p(\vec{q},\ \delta,\ c)$. The other is the number of required hash functions to be built, which is denoted as η_p. Then we present how to compute them.

Recall the definition of LSH, we must have $p_1' > p_2'$ so that the locality-sensitive hash function $h_i^r(\cdot)$ is useful in the ℓ_p space. By substituting p_1' and p_2', we have $p_2' = p(c, r_0 r/\delta^{\perp}) = p(c\delta^{\perp}/r, r_0) < p_1' \leq p_1 = p(1, r_0)$. Note that $p(s, r)$ is monotonically decreasing with s when r is fixed. Thus, we get $c\delta^{\perp}/r > 1$ as a necessary condition, which infers $r < c\delta^{\perp}$. Besides, we have $\delta^{\perp} \leq r \leq \delta^{\top}$ as we explained before. In summary, r must be chosen properly in the range of $[\delta^{\perp}, \min(\delta^{\top}, c\delta^{\perp})]$ in order for $p_1' > p_2'$ to hold. In details,

Algorithm 1: Sampling a point in $B_p(\vec{o}, 1)$ [13]

```
input  : dimensionality $d$, $\ell_p$ space
output: a random point $\vec{y}$ in $B_p(\vec{o}, 1)$, where $\vec{o}$ is the origin
1 Generate $d$ independent random real scalars $\xi_i \sim \tilde{G}(1, 1, p)$ ;
2 Construct a vector $\vec{x} \in \mathbb{R}^d$ of components $x_i = s_i \xi_i$, where $s_i$ is
  an independent random sign;
3 Construct $z = w^{1/d}$, where $w$ is a random variable uniformly
  distributed in the interval [0, 1] ;
4 return $\vec{y} = z \frac{\vec{x}}{\ell_p(\vec{x}, \vec{o})}$ ;
```

Algorithm 2: Calculating $Pr(\ell_1(\vec{o}, \vec{q}) \leq r | \ell_p(\vec{o}, \vec{q}) \leq 1)$

```
input  : dimensionality $d$, $\ell_p$ space, the number of sample points
         $n$, the number of buckets $b$
output: an array $p$ storing $Pr(\ell_1(\vec{o}, \vec{q}) \leq r | \ell_p(\vec{o}, \vec{q}) \leq 1)$ with
         different values of $r$
1  Initialize an array $r$ of $b$ dimensions recording the radii,
   $r[i] = \delta^{\perp} + (i+1) \times \frac{\min(\delta^{\top}, c\delta^{\perp}) - \delta^{\perp}}{b}$ for $0 \leq i < b$;
2  Initialize an array $c$ of $b$ dimensions to record the number of
   points in $B_1(\vec{o}, r)$, $c[i] \leftarrow 0$ for $0 \leq i < b$;
3  for $k \leftarrow 0$ to $n-1$ do
4      $\vec{v} \leftarrow$ a random point in $B_p(\vec{o}, 1)$;
5      Compute $\ell_1(\vec{o}, \vec{v})$ ;                /* $\vec{o}$ is the origin */
6      Find the minimal index $j$ such that $r[j] \geq \ell_1(\vec{o}, \vec{v})$;
7      for $i \leftarrow j$ to $b-1$ do
8          $c[i] \leftarrow c[i] + 1$;
9  for $i \leftarrow 0$ to $b-1$ do
10     $p[i] \leftarrow \frac{c[i]}{n}$;
11 return $p$;
```

r is a parameter for the functions of p'_1 and p'_2. p'_2 can be simply computed using Equation 4 when we build the base LSH index in the ℓ_1 space, while computing p'_1 is a nontrivial task. We begin with a lemma on the conditional probability.

LEMMA 3. *$Pr(\ell_s(\vec{x}, \vec{y}) \leq r | \ell_p(\vec{x}, \vec{y}) \leq \delta) =$ $Pr(\ell_s(\vec{u}, \vec{v}) \leq cr | \ell_p(\vec{u}, \vec{v}) \leq c\delta)$ for any $s, p, c > 0$.*

PROOF. See Appendix A.2. □

Based on Lemma 3, we can reduce the problem of calculating $Pr(e_4|e_2)$ to the computation of $Pr(\ell_1(\vec{o}, \vec{q}) \leq r \mid \ell_p(\vec{o}, \vec{q}) \leq 1)$, where $\delta = 1$ and $\delta^{\perp} \leq r < \min(\delta^{\top}, c\delta^{\perp})$. $Pr(\ell_1(\vec{o}, \vec{q}) \leq r | \ell_p(\vec{o}, \vec{q}) \leq 1)$ can be computed as follows:

$$Pr(\ell_1(\vec{o}, \vec{q}) \leq r | \ell_p(\vec{o}, \vec{q}) \leq 1) = \frac{Vol(B_1(\vec{q}, r) \bigcap B_p(\vec{q}, 1))}{Vol(B_p(\vec{q}, 1))}, \tag{15}$$

where $Vol(\cdot)$ outputs the volume of a given shape.

The challenge, however, is that computing the volume of $(B_1(\vec{q}, r) \bigcap B_p(\vec{q}, 1))$ for a random p and r exactly is very expensive if not impossible. Alternatively, we use the Monte Carlo method [9] to estimate $Pr(\ell_1(\vec{o}, \vec{q}) \leq r | \ell_p(\vec{o}, \vec{q}) \leq 1)$. Basically, this estimation is done by randomly sampling points in $B_p(\vec{q}, 1)$ and then calculating the percentage of the sampled points in $B_1(\vec{q}, r)$. Suppose we randomly sample n points in $B_p(\vec{q}, 1)$, and find m points are in $B_1(\vec{q}, r)$ as well. $\frac{m}{n}$ is roughly equal to $Pr(\ell_1(\vec{o}, \vec{q}) \leq r | \ell_p(\vec{o}, \vec{q}) \leq 1)$ if the number of samples is large enough.

$$Pr(\ell_1(\vec{o}, \vec{q}) \leq r | \ell_p(\vec{o}, \vec{q}) \leq 1) \approx \frac{m}{n}, \tag{16}$$

Note that the location of the center does not affect the probability. As a result, we sample points in $B_p(\vec{o}, 1)$, where $\vec{o}$ represents the origin. Given a d-dimensional space and a value of p, we can randomly sample points in $B_p(\vec{o}, 1)$ using Algorithm 1 [13]. In line 1, ξ_i is a random variable generated from a Generalized Gamma density function.

DEFINITION 7 (GENERALIZED GAMMA DENSITY [42]). *A random variable $x \in \mathbb{R}$ is generalized gamma distributed with three parameters $\alpha > 0$, $\lambda > 0$ and $\upsilon > 0$, denoted as $x \sim \tilde{G}(\alpha, \lambda, \upsilon)$, when x has the following density function:*

$$f(x) = \frac{\upsilon/\alpha^{\lambda}}{\Gamma(\lambda/\upsilon)} x^{\lambda-1} e^{-(x/\alpha)^{\upsilon}}, x \geq 0. \tag{17}$$

A random Generalized Gamma variable $x \sim \tilde{G}(1, \lambda, \upsilon)$ can be obtained from a Gamma random variable $z \sim G(\frac{\lambda}{\upsilon}, 1)$ as $x = z^{1/\upsilon}$ [42].

The formula involves the gamma function $\Gamma(t)$ $(t > 0)$, which is computed as:

$$\Gamma(t) = \int_0^{\infty} x^{t-1} e^{-x} dx, t > 0. \tag{18}$$

Therefore, we can sample points in $B_p(\vec{o}, 1)$ using Algorithm 1. Next we compute $Pr(\ell_1(\vec{o}, \vec{q}) \leq r | \ell_p(\vec{o}, \vec{q}) \leq 1)$ w.r.t. different values of r. As r is chosen in $[\delta^{\perp}, \min(\delta^{\top}, c\delta^{\perp})]$, we divide $[\delta^{\perp}, \min(\delta^{\top}, c\delta^{\perp})]$ into b buckets. Thus, the length of each bucket is ϕ $(\min(\delta^{\top}, c\delta^{\perp}) - \delta^{\perp})/b$. Afterwards, we set r to $(\delta^{\perp} + \phi)$, $(\delta^{\perp} + 2\phi)$, ..., $\min(\delta^{\top}, c\delta^{\perp})$ respectively and compute $Pr(\ell_1(\vec{o}, \vec{q}) \leq r | \ell_p(\vec{o}, \vec{q}) \leq 1)$, as described in Algorithm 2.

We initialize a counter to record the number of sample points that are located in $B_1(\vec{o}, r)$ for each radius r (line 2). Then, we randomly sample n points in $B_p(\vec{o}, 1)$. For each sampled point $\vec{v}$, we calculate its distance $\ell_1(\vec{v}, \vec{o})$ to $\vec{o}$ in the ℓ_1 space. For all the radii r greater than $\ell_1(\vec{v}, \vec{o})$, we know that the sample point $\vec{v}$ is inside $B_1(\vec{o}, r)$. Then we add one to the corresponding counters (lines 6-8). After sampling n points, we output $Pr(\ell_1(\vec{o}, \vec{q}) \leq r | \ell_p(\vec{o}, \vec{q}) \leq 1)$ for different values of r (lines 9-11). Algorithm 2 is an offline process. The approximation is more accurate if we sample more points and maintain more buckets. In the experiments, we set the number of sample points n to 1,000,000 and the number of buckets b to 1000.

Once we get the value of $Pr(\ell_1(\vec{o}, \vec{q}) \leq r | \ell_p(\vec{o}, \vec{q}) \leq 1)$, which is equal to $Pr(e_4|e_2)$, we can compute p'_1 and p'_2 as shown in Equations 13 and 14. In particular, $p(\cdot, \cdot)$ is computed as the one in Equation 4, because we build the base index in the ℓ_1 space. Figure 4 plots the values of p'_1 and p'_2 w.r.t. r for $\ell_{0.5}$ in $\mathbb{R}^{128}$ when the approximate ratio c is set to 2. The x axis represents the ratio of r to $\delta_{\perp}$, i.e. $(\frac{r}{\delta_{\perp}})$. As can be seen in the figure, p'_2 increases smoothly. In contrast, p'_1 increases slowly at the beginning. When the ratio reaches 1.4, p'_1 increases dramatically. When the ratio reaches around 1.55, p'_1 exceeds p'_2 and grows slowly at the end. We are interested in the cases when $p'_1 > p'_2$.

Recall that a (1, c, p_1, p_2)-sensitive hash function requires $p_1 > p_2$. In addition, the number of base hash functions η_p must be set to a certain value to ensure the correctness of the algorithm, which is related to $(p_1 - p_2)$ as shown in Equation 8 [21]. Equation 8 shows that the greater $(p_1 - p_2)$ is, the less base hash functions are required, resulting in less storage overhead. Therefore, we choose an optimal radius r, denoted as $\hat{r}$, by maximizing $(p'_1 - p'_2)$.

$$\hat{r} = \arg\max_{r} \ (p'_1 - p'_2) \tag{19}$$

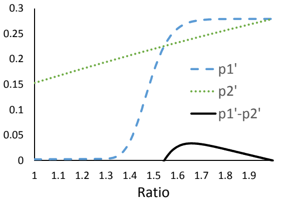


Figure 4: Values of p_1' and p_2' for $\ell_{0.5}$ in $\mathbb{R}^{128}$, $c = 2$

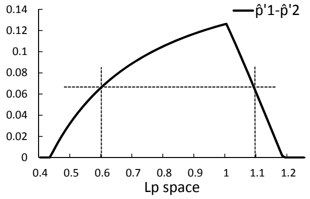


Figure 5: $(\hat{p}_1' - \hat{p}_2')$ w.r.t the ℓ_p spaces in $\mathbb{R}^{128}$, $c = 2$

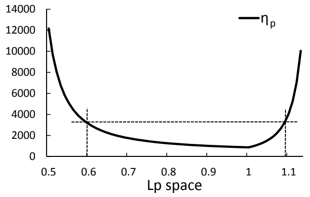


Figure 6: Number of hash functions required η_p in $\mathbb{R}^{128}$

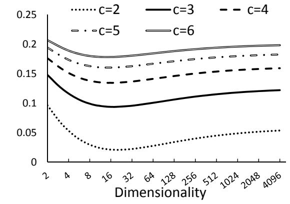


Figure 7: $(\hat{p}_1' - \hat{p}_2')$ w.r.t d in the $\ell_{0.5}$ space

For different ℓ_p spaces, the value of $\hat{r}$ varies. We pre-compute and save the values of $\hat{r}$ for different ℓ_p spaces, which is used when processing a query. Besides, we store the values of p_1' and p_2' when $r = \hat{r}$, which are denoted as $\hat{p}_1'$ and $\hat{p}_2'$ respectively. They are used to compute the number of required hash functions η_p when building the index and the collision count threshold θ_p in processing a query. Figure 5 plots $(\hat{p}_1' - \hat{p}_2')$ w.r.t. different ℓ_p spaces in $\mathbb{R}^{128}$, where the approximate ratio c is set to 2. When $p < 1$, $(\hat{p}_1' - \hat{p}_2')$ drops when p decreases. When $p < 0.44$, $\hat{p}_1'$ is always smaller than $\hat{p}_2'$, which indicates that the hash function built in the ℓ_1 space is no longer locality-sensitive in the corresponding ℓ_p space. When $p > 1$, $(\hat{p}_1' - \hat{p}_2')$ drops significantly when p increases. When $p > 1.18$, $\hat{p}_1'$ is always smaller than $\hat{p}_2'$.

Next, we calculate the number of required hash functions η for different ℓ_p spaces, which is computed as

$$\eta_p = \lceil \frac{\ln \frac{1}{\varepsilon}}{2(\hat{p}_1' - \hat{p}_2')^2}(1+z)^2 \rceil, \text{ where } z = \sqrt{\frac{\ln \frac{2}{\beta}}{\ln \frac{1}{\varepsilon}}}, \tag{20}$$

where ε and β are the same as the ones defined in Equation 8. Figure 6 plots η_p w.r.t. different ℓ_p spaces in $\mathbb{R}^{128}$, where $c = 2$, $\varepsilon = 0.01$ and $\beta = 0.0001$. When $p < 1$, η_p increases when p decreases, because η_p is inversely proportional to $(\hat{p}_1' - \hat{p}_2')$. Suppose we want to support a query $\mathcal{R}_{0.6}(\vec{q}, \delta, c)$ in the $\ell_{0.6}$ space. Correspondingly, at least $\eta_{0.6}$ hash functions are required to be materialized for the base index. Suppose we materialize $\eta_{0.6}$ hash functions as the base index. Using $\eta_{0.6}$ base hash functions, queries $\mathcal{R}_p(\vec{q}, r, c)$ can be answered in the ℓ_p spaces where $\eta_p \leq \eta_{0.6}$, as shown by the dashed line $(0.6 \leq p \leq 1.1)$ in Figure 6.

4. QUERY PROCESSING

In this section, we discuss how to leverage LazyLSH to process approximate range queries $(\mathcal{R}_p(\vec{q}, \delta, c))$ and nearest neighbor queries $(\mathcal{N}_p(\vec{q}, k, c))$ in different ℓ_p spaces.

4.1 Processing $\mathcal{R}_p(\vec{q}, \delta, c)$

Equation 20 indicates that we can find η_s and $\eta_{s'}$ with $s < s'$ and $\eta_s = \eta_{s'}$. Namely, two spaces share the same η value. This idea is also shown in Figure 5, where we get the same value of $|\hat{p}_1' - \hat{p}_2'|$ in $\ell_{0.6}$ and $\ell_{1.1}$. Suppose we have materialized η_s hash functions as the base index, where $0 < s < 1$. $\mathcal{R}_p(\vec{q}, \delta, c)$ queries can be answered in a series of ℓ_p spaces using the base index, where $s \leq p \leq s'$.

We use the $B_1(\vec{q}, \hat{r}\delta)$ ball in the ℓ_1 space to approximate the $B_p(\vec{q}, \delta)$ ball in the ℓ_p space as described in the previous section. We also pre-compute the corresponding $\hat{p}_1'$ and $\hat{p}_2'$ for the query ℓ_p space. To answer an $\mathcal{R}_p(\vec{q}, \delta, c)$ query, we modify the base hash function $h_i^*(\cdot)$ as $h_i^{\hat{r}\delta}(\cdot)$

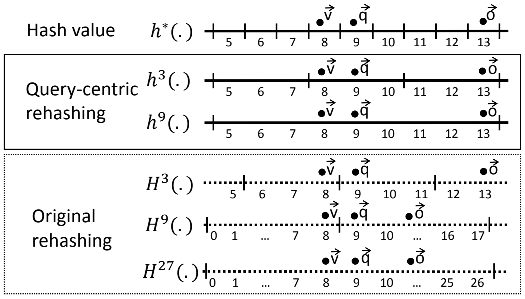


Figure 8: Query-centric rehashing

$$h_i^{\hat{r}\delta}(\vec{v}) = \lfloor \frac{h_i^*(\vec{v}) - (h_i^*(\vec{q}) \bmod \lfloor \hat{r}\delta \rfloor) + \lfloor \frac{3\hat{r}\delta}{2} \rfloor}{\hat{r}\delta} \rfloor \tag{21}$$

$(h_i^*(\vec{q}) \bmod \lfloor \hat{r}\delta \rfloor)$ can be viewed as a random integer as $\vec{q}$ is not known beforehand. Thus, $(\lfloor \frac{3\hat{r}\delta}{2} \rfloor - (h_i^*(\vec{q}) \bmod \lfloor \hat{r}\delta \rfloor))$ is a random positive integer, and $h_i^{\hat{r}\delta}(\cdot)$ is $(\hat{r}\delta, c\hat{r}\delta, p_1, p_2)$-sensitive in the ℓ_1 space. Based on Theorem 1, we know that $h_i^{\hat{r}\delta}(\cdot)$ is also $(\delta, c\delta, \hat{p}_1', \hat{p}_2')$-sensitive in the ℓ_p space if $\hat{p}_1' > \hat{p}_2'$.

Given a query $\mathcal{R}_p(\vec{q}, \delta, c)$ and the base hash functions, if an object collides with $\vec{q}$ more than θ_p times, the object is considered as a candidate. In particular, θ_p is defined as:

$$\theta_p = \frac{z\hat{p}_1' + \hat{p}_2'}{1 + z}\eta_p, \tag{22}$$

The $\mathcal{R}_p(\vec{q}, \delta, c)$ query is processed by retrieving the objects that are hashed to the same bucket as $\vec{q}$ for $h_i^{\delta\hat{r}}(\cdot)$. We adopt the virtual rehashing method [21]. If a point $\vec{v}$ has the same hash value as the query, i.e. $h_i^{\delta\hat{r}}(\vec{v}) = h_i^{\delta\hat{r}}(\vec{q})$, we can get the value range of $h_i^*(\vec{v})$ by expanding the equation.

$$h_i^*(\vec{q}) - \lfloor \frac{\delta\hat{r}}{2} \rfloor \leq h_i^*(\vec{v}) \leq h_i^*(\vec{q}) + \lfloor \frac{\delta\hat{r}}{2} \rfloor \tag{23}$$

From the above equation, we observe that the points hashed into the range of $[\ h_i^*(\vec{q}) - \lfloor \frac{\delta\hat{r}}{2} \rfloor,\ h_i^*(\vec{q}) + \lfloor \frac{\delta\hat{r}}{2} \rfloor\ ]$ in $h_i^*(\cdot)$ will collide with the query point which is mapped to the same bucket $h_i^{\delta\hat{r}}(\vec{q})$. We can see that the center of this range is the query point. Thus, we refer to the proposed hash function $h_i^{\hat{r}\delta}(\cdot)$ as the *query-centric rehashing* function.

Figure 8 presents the advantage of the proposed *query-centric rehashing* function. Suppose $\vec{q}$ is the query point. The first line represents the based hash function $h^*(\vec{q})$, where points $\vec{v}$, $\vec{q}$ and $\vec{o}$ are hashed to buckets 8, 9 and 13 respectively. The solid rectangle represents our proposed *query-centric rehashing* function for radius $r = 3, 9$, whereas the dashed rectangle represents the original rehashing function in Equation 7. For the *query-centric rehashing* function

$h^r(\cdot)$, $\vec{q}$ and $\vec{v}$ are hashed to the same bucket when $r = 3$, while $\vec{q}$ and $\vec{o}$ are hashed to the same bucket when $r = 9$. In contrast, for $H^r(\cdot)$, $\vec{q}$ and $\vec{o}$ are hashed to the same bucket when $r = 9$, while $\vec{q}$ and $\vec{v}$ are hashed to the same bucket when $r = 27$. We notice that the query point $\vec{q}$ is actually closer to $\vec{v}$ compared with $\vec{o}$, and thus $\vec{q}$ should collide with $\vec{v}$ in the same bucket first. Our proposed *query-centric rehashing* function can hold this property while $H^r(\cdot)$ cannot. In particular, $H^r(\cdot)$ might perform poorly in some cases, for example, when a query point is hashed to the bucket whose id is a multiple of the radius.

Algorithm 3 shows the pseudo code of processing an $\mathcal{R}_p(\vec{q}, \delta, c)$ query. An object with collision count larger than θ_p is considered as a candidate and its real distance to the query is computed. The algorithm stops until $\beta|\mathcal{D}|$ candidates are found. The algorithm is sound if the following two properties hold with a constant probability.

1. $\mathcal{P}'_1$: If $\vec{v} \in B_p(\vec{q}, \delta)$, then the number of $\vec{v}$'s collision with the query $\vec{q}$ is at least θ_p.

2. $\mathcal{P}'_2$: The total number of false positives is smaller than $\beta|\mathcal{D}|$, where $|\mathcal{D}|$ is the cardinality of a database $\mathcal{D}$.

Corollary 1 guarantees the correctness of the algorithm.

COROLLARY 1. *Given a error probability ε and a false positive rate β, if η_p is set as the one in Equation 20, we have $Pr(\mathcal{P}'_1) \geq (1 - \varepsilon)$ and $Pr(\mathcal{P}'_2) \geq 1/2$*

PROOF. This is a corollary of Lemma 1 in [21]. We refer readers to [21] for more details. □

4.2 Processing $\mathcal{N}_p(\vec{q}, k, c)$

The nearest neighbor query can be processed in a similar way. Algorithm 4 presents the general idea which can be viewed as the processing of a set of $\mathcal{R}_p(\vec{q}, \delta, c)$ queries with increasing radii. We initialize the starting radius to be $\frac{1}{\hat{r}}$ and issue an $\mathcal{R}_p(\vec{q}, \delta, c)$ query. If not enough results are found, we increase the radius to be $c\delta$ and issue a new range query. The process continues until enough results are returned.

We iteratively retrieve the points that are hashed into the range of $[\ h_i^*(\vec{q}) - \lfloor\frac{\delta\hat{r}}{2}\rfloor,\ h_i^*(\vec{q}) + \lfloor\frac{\delta\hat{r}}{2}\rfloor\ ]$ for $h_i^*(\cdot)$, because they collide with the query point for the modified hash function. However, as we have visited the points that are hashed in $[\ h_i^*(\vec{q}) - \lfloor\frac{\delta\hat{r}}{2c}\rfloor,\ h_i^*(\vec{q}) + \lfloor\frac{\delta\hat{r}}{2c}\rfloor\ ]$ in the last iteration when the radius is (δ/c), we skip those IDs (line 10).

The algorithm stops if (1) we have obtained k candidates whose ℓ_p distance to q is smaller than $c\delta$, or (2) we have found more than $k + \beta|\mathcal{D}|$ candidates with collision count larger than θ_p (lines 15-16). The stop conditions are defined based on $\mathcal{P}'_1$ and $\mathcal{P}'_2$ respectively. In particular, $\mathcal{P}'_1$ guarantees that a candidate will be found definitely, and $\mathcal{P}'_2$ ensures that there are no more than $\beta|\mathcal{D}|$ false positives. Therefore, we can stop the algorithm early and return the approximate kNNs of q in the ℓ_p space.

4.3 Multi-query Optimization

For the processing of queries under different ℓ_p metrics, the difference is that an object requires a different collision threshold to become a candidate. For fractional distance metrics, the smaller the p is, the larger the collision threshold is. This requires that more index entries are needed to be retrieved for the fractional distance metrics with a smaller p. The good news is that the index entries for a larger collision threshold (a smaller p) cover the index entries for a smaller collision threshold (a larger p), which means that no additional sequential I/O is required when we process queries in different ℓ_p spaces simultaneously for the same query point.

Algorithm 3: Answering $\mathcal{R}_p(\vec{q}, \delta, c)$

```
$T \leftarrow$ a hash table to record the collision count for each database object ;
$C \leftarrow$ a list to record the candidates ;
$\hat{r} \leftarrow$ the radius of $B_1(\vec{q}, \hat{r})$ to approximate $B_p(\vec{q}, 1)$ ;
for $i \leftarrow 0$ to $\eta_p - 1$ do
    Compute $h_i^{\delta\hat{r}}(\vec{q})$ ;
    Read the $IDs$ that are hashed in the range of
    $[\ h_i^*(\vec{q}) - \lfloor\frac{\delta\hat{r}}{2}\rfloor,\ h_i^*(\vec{q}) + \lfloor\frac{\delta\hat{r}}{2}\rfloor\ ]$ for $h_i^*(\cdot)$ ;
    foreach object $\vec{v} \in IDs$ do
        $T[\vec{v}] \leftarrow T[\vec{v}] + 1$ ;
        if $(T[\vec{v}] > \theta_p) \wedge (\vec{v} \notin C)$ then
            $C \leftarrow C \bigcup \{\vec{v}\}$ ;
            if $\ell_p(\vec{v}, \vec{q}) < c\delta$ then
                return $\vec{v}$ ;
            if $|C| > \beta|\mathcal{D}|$ then
                return NULL ;
```

Algorithm 4: Answering $\mathcal{N}_p(\vec{q}, k, c)$

```
1  $T \leftarrow$ a hash table to record the collision count for each database object ;
2  $C \leftarrow$ a list to record the candidates ;
3  $\hat{r} \leftarrow$ the radius of $B_1(\vec{q}, \hat{r})$ to approximate $B_p(\vec{q}, 1)$ ;
4  $\delta \leftarrow \frac{1}{\hat{r}}$ ;
5  while $TRUE$ do
6      for $i \leftarrow 0$ to $\eta_p - 1$ do
7          if $\delta = \frac{1}{\hat{r}}$ then
8              $IDs \leftarrow$ the points that are hashed to $h_i^*(\vec{q})$ ;
9          else
10             $IDs \leftarrow$ the points that are hashed in
               $[\ h_i^*(\vec{q}) - \lfloor\frac{\delta\hat{r}}{2}\rfloor, h_i^*(\vec{q}) - \lfloor\frac{\delta\hat{r}}{2c}\rfloor - 1\ ]$ or
               $[\ h_i^*(\vec{q}) + \lfloor\frac{\delta\hat{r}}{2c}\rfloor + 1, h_i^*(\vec{q}) + \lfloor\frac{\delta\hat{r}}{2}\rfloor\ ]$ for $h_i^*(\cdot)$ ;
11         foreach object $\vec{v} \in IDs$ do
12             $T[\vec{v}] \leftarrow T[\vec{v}] + 1$ ;
13             if $(T[\vec{v}] > \theta_p) \wedge (\vec{v} \notin C)$ then
14                 $C \leftarrow C \bigcup \{\vec{v}\}$ ;
15                 if $(|\{\vec{o}|\vec{o} \in C \wedge \ell_p(\vec{o}, \vec{q}) < c\delta\}| \geq k) \vee$
                   $(|C| > k + \beta|\mathcal{D}|)$ then
16                     return the $k$NNs in $C$ with the smallest $\ell_p$
                       distance to $\vec{q}$ ;
17     $\delta \leftarrow \delta * c$ ;
```

This finding motivates us to perform multiple queries under different ℓ_p metrics concurrently by sharing their I/Os. For example, suppose we need to answer queries $\mathcal{N}_p(\vec{q}, k, c)$ for p = 0.5, 0.6, 0.7, 0.8, 0.9 and 1.0. We can group them and answer them simultaneously. The I/O cost of processing these multiple queries is roughly the same as that of processing a single $\mathcal{N}_{0.5}(\vec{q}, k, c)$ query with additional random I/Os for retrieving candidates of other ℓ_p metrics.

5. EXPERIMENTS

In this section, we study the performance of LazyLSH with various datasets. We mainly focus on the following two issues: (1) How the index size changes with different parameter settings. (2) How LazyLSH performs with respect to the efficiency and effectiveness on various datasets.

5.1 Datasets and Queries

Table 3: Parameter Settings for the synthetic datasets

Notation	Description	Values
$\lvert\mathcal{D}\rvert$	Cardinality	100k, 200k, 400k, 800k, 1.6m
d	Dimensionality	100, 200, 400, 800, 1600
c	Approximate ratio	2, 3, 4, 5, 6
p	Supported ℓ_p space	0.5, 0.6, 0.7, 0.8, 0.9, 1.0

Table 4: Statistics of the real datasets and index sizes

Dataset	d	♯ points	value range	$\eta_{0.5}$	index size(MB)
Inria	128	4,455,041	[0, 255]	1358	23824
SUN	512	108,703	[0, 10,000]	916	1100
LabelMe	512	207,859	[0, 10,000]	959	2061
Mnist	784	60,000	[0, 255]	845	498

In the experiments, synthetic datasets and four real datasets are used: **Mnist**[2] [29], **Inria**[3] [27], **LabelMe**[4] [37] and **Sun**[5] [48]. The statistics of the synthetic datasets and real datasets are summarized in Tables 3 and 4 respectively. We refer readers to Appendix B.1 for more details of the datasets and query sets.

5.2 Evaluation Metrics

We follow the previous methods [45, 21, 22, 43] and adopt three metrics in our evaluations.

Space Consumption. The space is measured by the number of required hash tables and the index size.

Query Efficiency. The query efficiency is measured by the average number of I/Os of answering a query. If a block of an inverted list (4KB per block) is loaded into memory, the number of simulated I/Os (sequential) is increased by 1. If an object is visited to compute its distance to the query, the number of simulated I/Os (random) is increased by 1.

Overall Ratio. The overall ratio is defined as how many times farther a reported neighbor is compared to the real nearest neighbor. Formally, for an $\mathcal{N}_p(\vec{q}, k, c)$ query, $\{\vec{o_1}, \ldots, \vec{o_k}\}$ are the reported results sorted in ascending order of their distances to $\vec{q}$. Let $\{\vec{o_1^*}, \ldots, \vec{o_k^*}\}$ be the true kNNs sorted in ascending order of their distances to $\vec{q}$. The overall ratio is calculated as: $\frac{1}{k}\sum_{i=1}^{k} \ell_p(\vec{o_i}, \vec{q})/\ell_p(\vec{o_i^*}, \vec{q})$.

The I/O cost and the overall ratio are averaged over queries. Unless otherwise specified, we materialize $\eta_{0.5}$ hash functions for the index so that queries can be answered in the ℓ_p spaces, where $0.5 \le p \le 1$. By default, queries are issued in the $\ell_{0.5}$ space.

Competitors. To the best of our knowledge, LazyLSH is the first work of supporting approximate nearest neighbor queries on multiple distance functions with a single index. There is no direct competitor of our approach. Alternatively, we modify C2LSH [21] and SRS [43] as competitors.

- *C2LSH*: We build the index of C2LSH in the ℓ_1 space. Then we retrieve $(k + 100)$ candidates in the ℓ_1 space, and select the top-k points from the candidate set with the smallest ℓ_p distance to the query.

- *SRS*: We build the index of SRS in the ℓ_2 space because SRS uses the 2-stable distribution. We retrieve the candidates in the ℓ_2 space, and select the top-k points from the candidate set with the smallest ℓ_p distance to the query. The number of projected dimensions in SRS is set to 6 as the experimental setting in [43]. The approximate ratio c is set to 3 for comparison.

Implementation. Our algorithms were implemented in C++. All experiments were conducted on a PC with Intel Core i7-3770 CPU @ 3.40GHz, 8GB memory, 500GB hard disk, running Ubuntu 12.04LTS. The page size was set to 4 KB in the experiments.

5.3 Study on Synthetic Datasets

We use synthetic datasets to study the index size w.r.t. different parameter settings. As discussed in Section 3.3, the index size is affected by four parameters: (1) the cardinality of the dataset $|\mathcal{D}|$, (2) the dimensionality d, (3) the approximate ratio c, and (4) the range of supported ℓ_p spaces, where η_p hash functions are built for the base index. Table 3 shows the parameter settings of the synthetic datasets. The default values are underlined in the third column. We study the required index size w.r.t each parameter by varying one parameter and setting the other three parameters to the default values, as presented in Table 5.

Table 5: Index size w.r.t. different parameter settings

(a) Index size vs. the cardinality $|\mathcal{D}|$

$\lvert\mathcal{D}\rvert$	100k	200k	400k	800k	1.6m
$\eta_{0.5}$	923	979	1025	1071	1116
Size(MB)	1063	2211	4557	9250	18291

(b) Index size vs. the dimensionality d

d	100	200	400	800	1600
$\eta_{0.5}$	1223	1108	1025	966	879
Size(MB)	5011	4778	4557	4360	3997

(c) Index size vs. the approximate ratio c

c	2	3	4	5	6
$\eta_{0.5}$	7114	1025	570	425	355
Size(MB)	31609	4557	2531	1889	1577
I/Os	672184	77532	37252	24922	18712
Ratio	1.011	1.053	1.075	1.084	1.089

(d) Index size vs. the range of supported ℓ_p spaces

p	0.5	0.6	0.7	0.8	0.9	1.0
η_p	1025	711	579	507	462	432
Size(MB)	4557	3157	2576	2252	2051	1916

Effect of the cardinality $|\mathcal{D}|$: Table 5a shows the index size w.r.t $|\mathcal{D}|$. When $|\mathcal{D}|$ increases, the number of required hash functions increases, and so does the index size.

Effect of the dimensionality d: Table 5b shows the index size w.r.t d. It is worth noting that the index size decreases when d increases. This is because η_p changes with $(\hat{p}_1' - \hat{p}_2')$ as shown in Equation 20 and $(\hat{p}_1' - \hat{p}_2')$ varies w.r.t different numbers of dimensions. Figure 7 shows the relationship between the value of $(\hat{p}_1' - \hat{p}_2')$ and the dimensionality. The solid line in this figure plots $(\hat{p}_1' - \hat{p}_2')$ when the approximate ratio $c = 3$. As can be seen, $(\hat{p}_1' - \hat{p}_2')$ first decreases rapidly with the dimensionality and reaches the smallest value when $d = 16$. Then $(\hat{p}_1' - \hat{p}_2')$ increases slowly when the dimensionality increases. This explains the reason why the index size decreases with d when $d > 100$ for the synthetic datasets.

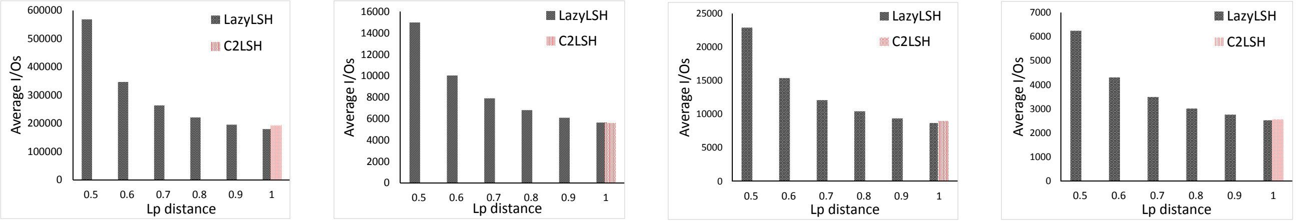
Effect of the approximate ratio c: Table 5c shows the index size w.r.t c. When c increases, the index size decreases. The reason can be also explained by Figure 7. As shown in

[2] http://yann.lecun.com/exdb/mnist/

[3] http://lear.inrialpes.fr/~jegou/data.php

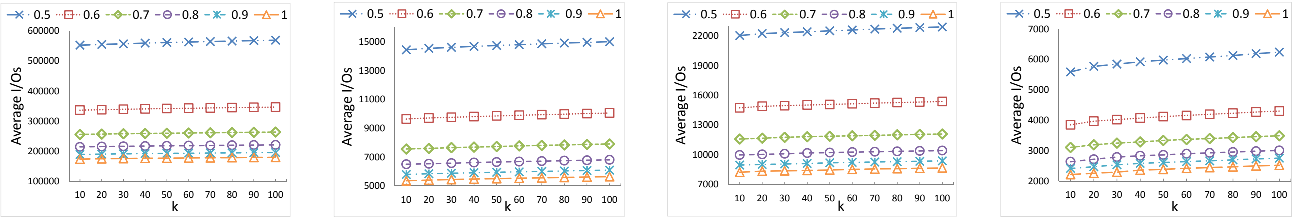
[4] http://labelme.csail.mit.edu/Release3.0/

[5] http://sundatabase.mit.edu/



(a) Inria (b) SUN (c) LabelMe (d) Mnist

Figure 9: I/O costs on real datasets w.r.t. the ℓ_p distance



(a) Inria (b) SUN (c) LabelMe (d) Mnist

Figure 10: I/O costs on real datasets w.r.t. the number of k

the figure, for a fixed dimension, $(\hat{p}'_1 - \hat{p}'_2)$ increases when c increases, which leads to the smaller index size. When $c = 2$, the number of required hash functions $\eta_{0.5}$ is around seven times of $\eta_{0.5}$ for $c = 3$.

In addition, we test the number of I/Os and the overall ratio when processing approximate queries with different approximate ratio c. We notice that: (1) the number of I/Os decreases significantly as c increases. The number of I/Os when $c = 2$ is about nine times of that when $c = 3$, because of the difference in index size. (2) the overall ratio increases with c because a larger approximation ratio is applied, which means that we can get more accurate results when we use a smaller c. Therefore, c can be viewed as a parameter to be set as a trade-off between query accuracy and efficiency (index size). To make it comparable to C2LSH, which used $c = 2$ or 3 in its experiment [21], we set $c = 3$ for LazyLSH in the rest of the experiments.

Effect of the range of supported ℓ_p spaces: Table 5d shows the index size w.r.t the range of supported ℓ_p spaces. To support a larger range of ℓ_p spaces, we need to materialize more hash functions. For instance, we need 2.37x hash functions to support queries in a range of ℓ_p spaces, where $0.5 \leq p \leq 1$, compared to the number of hash functions required for the single ℓ_1 space.

5.4 Study on Real Datasets

5.4.1 *Index Size*

We first study the index size for the real datasets. We materialize $\eta_{0.5}$ hash functions as the base index so that queries $\mathcal{R}_p(\vec{q}, \delta, c)$ can be supported in a range of ℓ_p spaces, where $0.5 \leq p \leq 1$. Table 4 shows the number of hash functions required and the index size for the real datasets. When the dimensionality increases, fewer hash functions are required. This observation is consistent with the result of the synthetic datasets shown in Table 5b.

5.4.2 *I/O Costs of Processing Queries*

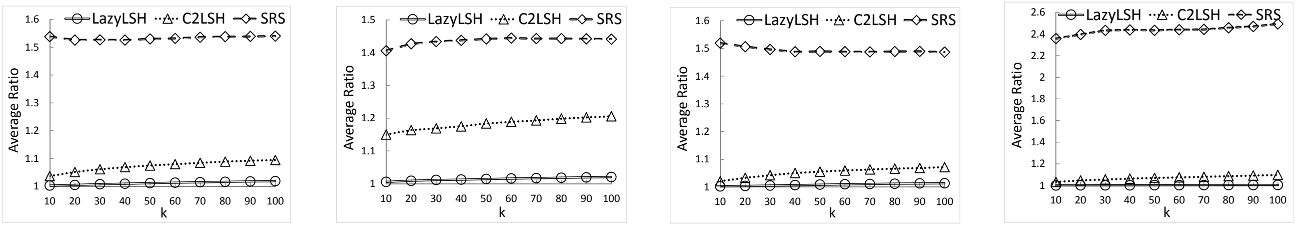
Next we study the performance of LazyLSH in terms of the I/O cost of processing a query.

I/O Costs w.r.t. the query ℓ_p space: Figure 9 plots the average I/Os of processing a query w.r.t. the ℓ_p space where the number of nearest neighbors k is set to 100. The query processing in the $\ell_{0.5}$ space incurs more I/O overhead than the ℓ_1 space. Generally, a smaller p will lead to a higher I/O cost. This is because the processing of queries in the $\ell_{0.5}$ space requires a higher collision threshold for an object to become a candidate and more index entries are required to be read. In the ℓ_1 space, the performance of LazyLSH is similar to C2LSH. The average I/O costs for these two methods are at the same level. However, C2LSH can only support queries in the ℓ_1 space, while in contrast, LazyLSH is designed to support queries in a larger range of ℓ_p distances. Please note that the I/O cost of SRS is not reported as the provided implementation[6] is in-memory and SRS is based on the ℓ_2 distance. Still, it is obvious that SRS can achieve the best performance in terms of the I/O cost as its index size is one order of magnitude smaller than that of C2LSH as reported in [43]. Even though SRS has a small I/O cost, its reliance on the 2-stable distribution in the ℓ_2 space constrains us from using its technique as the base index structure. We refer readers to Appendix C for more details.

I/O Costs w.r.t. the number of k: Figure 10 plots the average I/Os of processing a query w.r.t. the number of returned nearest neighbors. The horizontal axis k represents the number of returned nearest neighbors, ranging from 10 to 100. We observe that there is a slight increase of I/Os on all the four datasets when k increases. This indicates that users can issue a query with a larger k to get more precise nearest neighbors with a few additional I/Os.

I/O Costs w.r.t. the multiple-query optimization: One application of using LazyLSH is to select the optimal ℓ_p metric, which is shown to be highly application-dependent [20]. One way is to use classifications to select the best ℓ_p

[6] https://github.com/DBWangGroupUNSW/SRS



(a) Inria

(b) SUN

(c) LabelMe

(d) Mnist

Figure 11: Average overall ratios on real datasets

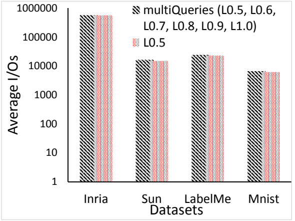


Figure 12: Multiple-query optimization

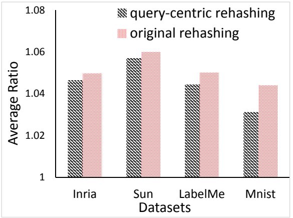


Figure 13: Impact of query-centric rehashing

metric. In particular, we retrieve the approximate kNNs in different ℓ_p spaces. Then we select the optimal ℓ_p metric with the highest classification accuracy as shown in Table 1. This approach requires performing approximate kNN queries in different ℓ_p spaces, which can be processed as the multi-query optimization in Section 4.3.

Figure 12 presents the I/O cost of processing a single query versus that of processing multiple queries concurrently. We try to find the kNNs of the same data point in different ℓ_p spaces (six queries in the $\ell_{0.5}$, $\ell_{0.6}$, $\ell_{0.7}$, $\ell_{0.8}$, $\ell_{0.9}$ and ℓ_1 spaces in this experiment). The queries of different ℓ_p spaces are processed together by sharing their I/Os, as they all probe many common hash buckets. As indicated in the figure, processing multiple queries concurrently only incurs a few more I/Os than processing a single query. This observation motivates us to process queries of different ℓ_p distances in a batch, instead of processing them individually. With the help of the multiple-query optimization, we can easily get the approximate kNNs of different ℓ_p metrics and choose the optimal ℓ_p metric for a particular dataset based on classification methods such as the kNN classifier as presented in Table 1 in the introduction. We also compare the running time of LazyLSH with that of the linear scan method, which is presented in Appendix B.2.

5.4.3 *Overall Ratio*

Overall Ratio for queries in fractional metrics: We then compare LazyLSH with the existing work in the context of the overall ratio. Figure 11 presents the average overall ratio for queries in the $\ell_{0.5}$ space. In general, LazyLSH outperforms C2LSH in terms of the overall ratio. In most cases, the overall ratio of LazyLSH is less than 1.02. In contrast, C2LSH does not perform well for queries in the $\ell_{0.5}$ space, because C2LSH is designed to answer queries in the ℓ_1 or ℓ_2 spaces and it is not optimized to answer queries for fractional distances. Therefore, LazyLSH has better performance in terms of the overall ratio.

Impact of the query-centric rehashing: Next we study the impact of the query-centric rehashing. To achieve a fair comparison, the queries are conducted in the same index with different rehashing methods. In particular, the queries are conducted in the ℓ_1 space and the approximate 100 NNs are returned. Figure 13 plots the average overall ratio for different rehashing methods. We notice that our query-centric rehashing method outperforms the original rehashing method on all the four datasets. This is because the query-centric rehashing method is likely to retrieve nearby points, while the original rehashing method may retrieve distant points in some cases as stated in Figure 8.

6. RELATED WORK

6.1 Similarity Search and kNN Processing

The similarity search problem is of fundamental importance to a variety of applications such as classification [17], clustering [35], semi-supervised learning [52], semi-lazy learning [51, 50], collaborative filtering [39] and near-duplicate detection [8]. Given a query object $\vec{q}$ represented as a high-dimensional vector, a typical similarity search retrieves the k-nearest neighbors (kNNs) of $\vec{q}$ using a specific distance function.

kNN search has been well studied in low-dimensional spaces [36, 38]. However, retrieving exact results becomes a nontrivial problem when the dimensionality increases due to the "curse of dimensionality". It has been shown that the average query time with an optimal indexing scheme is superpolynomial with the dimensionality [34]. When the dimensionality is sufficiently large, conventional kNN search approach becomes even slower than the linear scan approach [18]. To address the problem, approximate nearest neighbor search is introduced as an alternative solution, which trades off the accuracy to speed up the search. One typical solution is to employ the Locality-Sensitive Hashing (LSH) because of its precise theoretical guarantees and empirical performance.

6.2 Locality-Sensitive Hashing

LSH is first introduced by Indyk and Motwani [26]. The basic idea of LSH is that an LSH function produces a hash bit of a data point by projecting the data point to a random hyperplane. Statistics guarantees that the nearby points in the projected hyperplane are likely to be close to each other in the original space. In addition, multiple hash tables are built independently, aiming to enlarge the probability that similar data points are mapped to similar hash codes. Several LSH families have been discovered for metric distances [4], including the Hamming distance between binary vectors [26], the Jaccard distance between sets [11, 10], the arccos distance measuring the angles between vectors [14], the Manhattan distance [3, 18], and the Euclidean distance [18].

For the ℓ_p metrics, E2LSH [18] is the first LSH method that supports approximate nearest neighbor queries in the Euclidean space. The main drawback of E2LSH is that it needs to build multiple indexes with different radii, resulting in high maintenance overhead. To address the issue of high storage costs, multi-probe LSH [32] is proposed. It not only checks the data points that fall in the same bucket as the query point, but also probes multiple buckets that are likely to contain results in a hash table, which leads to the use of fewer hash tables. However, it still suffers from the need for building hash tables at different radii.

LSB-tree [45] is the first work that does not need to build hash tables at different radii. It groups hash values as a one-dimensional Z-order value and indexes it in a B-tree. Index entries for different radii can be retrieved in corresponding ranges in the B-tree. In addition, an LSB forest with multiple trees can be built to improve the search performance. C2LSH [21] further improves the LSB-tree method by introducing techniques of collision counting and virtual rehashing. It uses one hash function per hash table so that the number of hash buckets to read does not increase exponentially when the query radius increases. Furthermore, SRS [43] projects data points from the original high-dimensional space into a low-dimensional space via 2-stable projections. The major observation is that the ℓ_2 distance in the projected space over the ℓ_2 distance in the original space follows a chi-squared distribution, which has a sharp concentration bound. Therefore, SRS can index the data points in the low-dimensional projected space and use the chi-squared distribution to perform queries. In this case, the index size is significantly reduced.

LSH variants are proposed to address the drawbacks of the traditional LSH methods. Traditional LSH methods suffer from the disadvantage of generating a large number of the false positives. To address this problem, BayesLSH [40] is proposed. It integrates the Bayesian statistics and performs candidate pruning and similarity estimation using LSH. It can quickly prune away false positives, which leads to a significant speedup. Traditional LSH methods also suffer from the disadvantage of accessing many candidates, which brings a larger number of random I/Os. To address this, SortingKeys-LSH [30] is presented to order the compound hash keys so that the data points are sorted accordingly in the index to reduce the I/O cost.

6.3 Fractional Distance Metric

Euclidean distance [15, 23, 41, 6, 36] is widely used in kNN search. However, it was shown that when the dimensionality is high, the Euclidean distance introduces the concentration problem [7]. Namely, the distance between any random pair of high dimensional data points is almost identical. Therefore, the effectiveness of the Euclidean distance in the high-dimensional space is not clear [24, 1].

To address the concentration problem, one direction is to consider partial similarities, which have been drawn increasing attention in the last decade [28, 46, 12, 44]. Tung et al. [46] introduced the k-n-match model to discover the partial similarity in n dimensions, where n is a given integer smaller than dimensionality and these n dimensions are determined dynamically to make the query and the returned results be the most similar. It has been shown that the k-n-match model yields better results than the traditional kNN query in identifying similar objects by partial similarities.

Another direction is to investigate different distance metrics. Aggarwal et al. [24, 1] examined the "curse of dimensionality" problem from the perspective of distance metrics. They found that the Manhattan distance (ℓ_1) metric is more effective than the Euclidean distance (ℓ_2) metric in the high-dimensional space. They further introduced and examined the fractional distance (ℓ_p for $0 < p < 1$) metrics, which are less concentrated. It was shown that the fractional distance metrics provide more meaningful results from both the theoretical and empirical perspectives. Their experimental results verified that fractional distance metrics improve the effectiveness of standard clustering algorithms.

Later, fractional distance metrics have been applied to applications such as content-based image retrieval [25, 47]. The experiments showed that the performances of fractional distance metrics outperform the ℓ_1 and ℓ_2 metrics. In particular, the $\ell_{0.5}$ distance consistently outperforms both ℓ_1 and ℓ_2 metrics in their experiments. Still, the experimental results showed that the optimal ℓ_p metric is application-dependent and required to be learned for each dataset.

Recently, it was argued that the analysis of the concentration phenomenon is based on the assumption of independent and identically distributed variables [20], which might not be true in real datasets. The authors showed that the optimal ℓ_p metric is actually application-dependent. In order to explore the high-dimensional data, it is important to support different distance metrics. Hence, users can try and select the best ℓ_p metric for their applications. However, to the best of our knowledge, none of the existing LSH approaches can support multiple distance metrics. LazyLSH is the first work that supports approximate kNN processing in multiple fractional metrics using a single index. In this paper, we use the index structure of C2LSH as the base index structure. LazyLSH can also be used other LSH index structures as the base index, which is presented in Appendix C.

7. CONCLUSIONS

In this paper, we proposed an efficient mechanism called LazyLSH to answer approximate kNN queries under fractional distance metrics in high-dimensional spaces. We observed that an LSH function in a specific ℓ_p space can be extended to support queries in other spaces. Based on this observation, we materialized a base LSH index in the ℓ_1 space and used it to process approximate kNN queries in different ℓ_p spaces. We conducted extensive experiments on synthetic and real datasets, and the experimental results showed that LazyLSH provides accurate results and improves existing machine learning algorithms for retrieving approximate kNN under different fractional distance metrics.

8. ACKNOWLEDGEMENT

The work by Anthony K.H. Tung was partially supported by NUS FRC Grant R-252-000-370-112. The work by Sai Wu was partially supported by the National Basic Research Program of China 973 (No. 2015CB352400). This research was carried out at the NUS-ZJU Sensor-Enhanced social Media (SeSaMe) Centre. It is supported by the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the Interactive Digital Media Programme Office.

9. REFERENCES

[1] C. Aggarwal, A. Hinneburg, and D. Keim. On the surprising behavior of distance metrics in high dimensional space. In *ICDT*, volume 1973, pages 420–434. 2001.

[2] N. Ailon and B. Chazelle. Approximate nearest neighbors and the fast johnson-lindenstrauss transform. In *STOC*, pages 557–563, 2006.

[3] A. Andoni and P. Indyk. Efficient algorithms for substring near neighbor problem. In *SODA*, pages 1203–1212, 2006.

[4] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *COMMUNICATIONS OF THE ACM*, 51(1):117–122, 2008.

[5] S. Arya and D. M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *SODA*, pages 271–280, 1993.

[6] N. Beckmann, H.-P. 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.

[7] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is ąřnearest neighborąś meaningful? In *ICDT*, pages 217–235. 1999.

[8] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *SIGKDD*, pages 39–48, 2003.

[9] K. Binder and D. Heermann. *Monte Carlo simulation in statistical physics: an introduction*. Springer Science & Business Media, 2010.

[10] A. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29, 1997.

[11] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. In *WWW*, pages 1157–1166, 1997.

[12] A. M. Bronstein, M. M. Bronstein, A. M. Bruckstein, and R. Kimmel. Partial similarity of objects, or how to compare a centaur to a horse. *International Journal of Computer Vision*, 84(2):163–183, 2008.

[13] G. Calafiore, F. Dabbene, and R. Tempo. Uniform sample generation in lp balls for probabilistic robustness analysis. In *CDC*, volume 3, pages 3335–3340, 1998.

[14] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002.

[15] J. G. Cleary. Analysis of an algorithm for finding nearest neighbors in euclidean space. *ACM Trans. Math. Softw.*, 5(2):183–192, 1979.

[16] G. Cormode, P. Indyk, N. Koudas, and S. Muthukrishnan. Fast mining of massive tabular data via approximate distance computations. In *ICDE*, pages 605–614, 2002.

[17] T. Cover and P. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Trans. on*, 13(1):21–27, 1967.

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

[19] M. Douze, H. Jégou, H. Sandhawalia, L. Amsaleg, and C. Schmid. Evaluation of gist descriptors for web-scale image search. In *CIVR*, pages 19:1–19:8, 2009.

[20] D. Francois, V. Wertz, and M. Verleysen. The concentration of fractional distances. *TKDE*, 19(7):873–886, 2007.

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

[22] J. Gao, H. V. Jagadish, W. Lu, and B. C. Ooi. Dsh: Data sensitive hashing for high-dimensional k-nnsearch. In *SIGMOD*, pages 1127–1138, 2014.

[23] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[24] A. Hinneburg, C. C. Aggarwal, and D. A. Keim. What is the nearest neighbor in high dimensional spaces? In *VLDB*, pages 506–515, 2000.

[25] P. Howarth and S. Rüger. Fractional distance measures for content-based image retrieval. In *ECIR*, pages 447–456, 2005.

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

[27] H. Jegou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *ECCV*, pages 304–317, 2008.

[28] L. J. Latecki, R. Lakaemper, and D. Wolter. Optimal partial shape similarity. *Image and Vision Computing*, 23(2):227 – 236, 2005. Discrete Geometry for Computer Imagery.

[29] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[30] Y. Liu, J. Cui, Z. Huang, H. Li, and H. T. Shen. Sk-lsh: An efficient index structure for approximate nearest neighbor search. *PVLDB.*, 7(9):745–756, 2014.

[31] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, 2004.

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

[33] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 2, 2009.

[34] V. Pestov. Lower bounds on performance of metric tree indexing schemes for exact similarity search in high dimensions. *Algorithmica*, 66:310–328, 2013.

[35] D. Ravichandran, P. Pantel, and E. Hovy. Randomized algorithms and nlp: Using locality sensitive hash function for high speed noun clustering. In *ACL*, pages 622–629, 2005.

[36] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *SIGMOD*, pages 71–79, 1995.

[37] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. Labelme: A database and web-based tool for image annotation. *Int. J. Comput. Vision*, 77(1-3):157–173, 2008.

[38] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers Inc., 2005.

[39] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, pages 285–295, 2001.

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

[41] T. K. Sellis, N. Roussopoulos, and C. Faloutsos. The r+-tree: A dynamic index for multi-dimensional objects. In *VLDB*, pages 507–518, 1987.

[42] E. W. Stacy. A generalization of the gamma distribution. *The Annals of Mathematical Statistics*, pages 1187–1192, 1962.

[43] 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.

[44] X. Tan, S. Chen, Z.-H. Zhou, and J. Liu. Face recognition under occlusions and variant expressions with partial similarity. *Information Forensics and Security, IEEE Transactions on*, 4(2):217–230, 2009.

[45] Y. Tao, K. Yi, C. Sheng, and P. Kalnis. Efficient and accurate nearest neighbor and closest pair search in high-dimensional space. *ACM Trans. Database Syst.*, 35(3):20:1–20:46, 2010.

[46] A. K. H. Tung, R. Zhang, N. Koudas, and B. C. Ooi. Similarity search: A matching based approach. In *VLDB*, pages 631–642, 2006.

[47] H. Wang, S. Zhang, W. Liang, F. Wang, and Y. Yao. Content-based image retrieval using fractional distance metric. In *IASP*, pages 1–5, 2012.

[48] J. Xiao, J. Hays, K. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *CVPR*, pages 3485–3492, 2010.

[49] M.-L. Zhang and Z.-H. Zhou. Ml-knn: A lazy learning approach to multi-label learning. *Pattern Recognition*, 40(7):2038 – 2048, 2007.

[50] J. Zhou and A. K. Tung. Smiler: A semi-lazy time series prediction system for sensors. In *SIGMOD*, pages 1871–1886, 2015.

[51] J. Zhou, A. K. Tung, W. Wu, and W. S. Ng. A ąřsemi-lazyąś approach to probabilistic path prediction in dynamic environments. In *SIGKDD*, pages 748–756, 2013.

[52] X. Zhu and A. B. Goldberg. Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning*, 3(1):1–130, 2009.

APPENDIX

A. PROOF

A.1 Proof of Lemma 2

Proof. We know:

$$p(cs, cr) = \int_0^{cr} \frac{1}{cs} f_p(\frac{t}{cs})(1 - \frac{t}{cr})dt$$

Let

$$x = t/c$$

.

Then we get:

$$\begin{aligned} p(cs, cr) &= \int_0^{r} c\frac{1}{cs} f_p(\frac{cx}{cs})(1 - \frac{cx}{cr})dx \\ &= \int_0^{r} \frac{1}{s} f_p(\frac{x}{s})(1 - \frac{x}{r})dx = p(s, r) \quad \square \end{aligned}$$

A.2 Proof of Lemma 3

Proof. For any $\vec{x}, \vec{y} \in \mathbb{R}^d$, let $\vec{u} = c\vec{x}$ and $\vec{v} = c\vec{y}$. Then, we have

$$\begin{aligned} \ell_s(\vec{u}, \vec{v}) =& (\sum_{i=1}^{d} (u_i - v_i)^s)^{1/s} \\ =& (\sum_{i=1}^{d} (cx_i - cy_i)^s)^{1/s} \\ =& c\ \ell_s(\vec{x}, \vec{y}) \end{aligned}$$

Therefore, we get $\ell_s(\vec{x}, \vec{y}) \leq r \iff \ell_s(\vec{u}, \vec{v}) \leq cr$. Similarly, $\ell_p(\vec{x}, \vec{y}) \leq \delta \iff \ell_p(\vec{u}, \vec{v}) \leq c\delta$. Note $\vec{u}$ and $\vec{x}$, $\vec{v}$ and $\vec{y}$ are one-to-one corresponding. Thus, we get

$$Pr(\ell_s(\vec{x}, \vec{y}) \leq r | \ell_p(\vec{x}, \vec{y}) \leq \delta) = Pr(\ell_s(\vec{u}, \vec{v}) \leq cr | \ell_p(\vec{u}, \vec{v}) \leq c\delta)$$

□

B. MORE ON EXPERIMENTS

B.1 Datasets and Query sets

Synthetic datasets: We use synthetic datasets to study the influences of the dimensionality and the cardinality. We first fix the cardinality to be 40,000 and generate datasets with different dimensionality as {100, 200, 400, 800, 1600}. Then we fix the dimensionality to be 400 and generate datasets with different cardinality as {100k, 200k, 400k, 800k, 1.6m}. The value of each dimension is an integer randomly chosen from [0, 10000]. For each synthetic dataset, we randomly generate 50 query points as a query set.

Inria: The Inria dataset contains 1,491 holiday photos. 4,455,091 SIFT features [31] are extracted from the images, and each feature is represented as a 128-dimensional point. The value of each dimension is an integer in the range of [0, 255]. We randomly select 50 feature points as our query set and remove those features from the dataset during the query processing to avoid returning the same feature.

SUN: SUN is a dataset containing 108,753 images, each of which is attached with a class label to indicate which scene category the image belongs to. We obtain the GIST feature [19] of each image and generate a corresponding 512-dimensional data point. The value of each dimension is normalized to be an integer in the range of [0, 10,000]. We randomly pick 50 data points as a query set.

LabelMe: LabelMe is a dataset containing 207,909 images. We apply the processing method of SUN to the LabelMe dataset. Hence, the size of LabelMe is 207,859.

Mnist: Mnist is a dataset consisting of 60,000 pictures of handwritten digits. Each picture has a class label representing which digit the picture shows. Each picture is represented as a 784-dimensional point, of which each dimension is an integer ranging from 0 to 255. In addition, Mnist contains a test set of 10,000 points. We randomly choose 50 points to form a query set.

B.2 Additional experiments

This experiment analyzes the query time of LazyLSH versus that of the linear scan method. As presented in Table 1 and Section 5.4.2, we need to retrieve approximate kNNs in different ℓ_p spaces to test the accuracy of the kNN classifier in order to pick the optimal ℓ_p metric.

We first study the effect of the multi-query optimization in terms of query time. We use the synthetic dataset with cardinality of 400k and dimensionality of 400 in this experiment. We set the approximate ratio $c = 3, 4, 5, 6$ and the number of returned points $k = 100$. The result for $c = 2$ is skipped as the I/O cost for $c = 2$ is much higher than others as shown in Table 5c. The single query is performed in the $\ell_{0.5}$ space, and the multiple queries are preformed in the $\ell_{0.5}$, $\ell_{0.6}$, $\ell_{0.7}$, $\ell_{0.8}$, $\ell_{0.9}$ and ℓ_1 spaces using our multi-query optimization. Figure 14 shows that the average query time of the two methods. The query time of LazyLSH with $c = 4$ for the single query is at the same level as that of the linear scan. However, the running time of the linear scan method increases dramatically when multiple queries are answered. In contrast, the running time of LazyLSH for processing multiple queries remains at the same level as that for performing the single query. This finding is consistent to the result in Figure 12, because the number of I/Os increases by a small amount when processing multiple queries in a batch.

In addition, Figure 14 shows that the running time decreases as the approximate ratio c increases. In fact, the choice of c can be viewed as a trade-off between accuracy and query time. As it is shown in Figure 11, LazyLSH achieves the highest accuracy among the LSH competitors in performing approximate queries using fractional distance metrics. We are interested in how the accuracy changes given different settings of c. Figure 15 presents the average overall ratio of performing queries w.r.t c in different ℓ_p spaces. The overall ratio is fairly well. Even for $c = 6$, the overall ratio

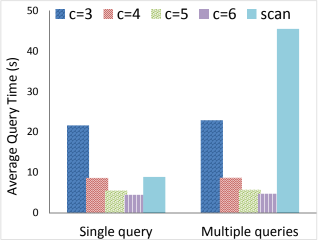


Figure 14: Query time with multi-query optimization

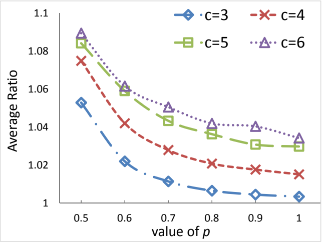


Figure 15: Average overall ratios w.r.t. c

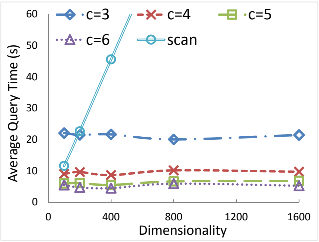


Figure 16: Average query time w.r.t. dimensionality

in the $\ell_{0.5}$ space is still smaller than 1.1. This finding indicates that we can set c to be larger for faster speed while the accuracy remains acceptable. We find that $c = 5$ or 6 might be good choices considering the trade-off between accuracy and query time.

Next we study the query time with respect to the dimensionality. We use the synthetic datasets with cardinality of 400k and dimensionality of 100, 200, 400, 800 and 1600. Figure 16 presents the average running time for processing multiple queries with different dimensionality. The average running time of the linear scan method increases linearly with the dimensionality because the CPU time of computing the distance to the query grows linearly. Please note that the results of the linear scan for $d = 800$ and 1600 are not shown for better viewing of the results of LazyLSH, and the results follow the trend. In contrast, the query time of LazyLSH remains at the same level for different dimensionality. When the approximate ratio is greater than or equal to 4, LazyLSH achieves better results in terms of query time compared to the linear scan method. In addition, LazyLSH gains more speedup when the number of dimensions increases, because the number of required hash functions does not increase linearly as presented in Table 5b. Again, the running time decreases given a larger approximate ratio c. In order to balance the accuracy and query time, we can use a larger c for those datasets with low dimensionality, and use a smaller c for those datasets with high dimensionality.

C. EXTENDING LAZYLSH TO OTHER EXISTING METHODS

LazyLSH is orthogonal to most previous work on LSH. LazyLSH targets at answering approximate nearest neighbor queries in different ℓ_p spaces using a single index. In this paper, the same index structure of C2LSH is used to process approximate nearest neighbor queries. Besides C2LSH, LazyLSH can also adopt other index structures. For instance, LazyLSH can be easily combined with the E2LSH structure. The difference is how to choose the optimal value of $\hat{r}$ in Equation 19. To fit the E2LSH structure and its variants [18, 30], we can choose $\hat{r}$ with a different optimization method. As stated in [18], $\rho = \frac{\ln 1/p'_1}{\ln 1/p'_2}$ is a parameter to be minimized in the algorithm. The smaller ρ is, the more precise results are returned. Therefore, we choose a radius r such that ρ is minimized, formally,

$$\hat{r} = \arg\min_r \rho = \arg\min_r \frac{\ln 1/p'_1}{\ln 1/p'_2}. \tag{24}$$

SRS [43] proposed a tiny index to support approximate nearest neighbor queries in the ℓ_2 space. It is reported that the index size of SRS is at least one order of magnitude smaller than that of C2LSH [21]. SRS projects data points from the original high-dimensional space into a low-dimensional space via 2-stable projections. The major observation is that the square of the ratio of the ℓ_2 distance between two points in the projected space to the ℓ_2 distance between them in the original space follows the standard chi-squared distribution. Therefore, SRS can index the data points in the low-dimensional projected space and process approximate nearest neighbor queries for high-dimensional points with theoretical guarantees based on the chi-squared distribution. However, such 2-stable projection restricts SRS to building its index in the ℓ_2 space. In contrast, our LazyLSH method is proposed based on the intuition that given $p > 0$ and $s > 0$, if two points are close in the ℓ_p space, then they are likely to be close in the ℓ_s space as well. Let $\epsilon = |p - s|$. The property holds with a higher probability for a smaller ϵ. In order to better support the approximate queries in fractional spaces, we materialize the base index in the ℓ_1 space.

To test whether the index built in the ℓ_2 space can answer approximate queries in fractional spaces, we try to use an ℓ_2 ball to approximate an $\ell_{0.5}$ ball. We set the approximate ratio $c = 3$. Our experimental result showed $p'_1 < p'_2$ when the dimensionality is greater than five, which means that the LSH functions in the ℓ_2 space might not necessarily be locality-sensitive in the $\ell_{0.5}$ space when the dimensionality increases. Based on the analysis above, we conclude that it is hard to integrate SRS into LazyLSH to support queries in fractional distances because SRS has its restriction on building its index in the ℓ_2 space. However, if there is a subsequent method which can build a tiny index in the ℓ_1 space, it should be easy to extend LazyLSH to the method.