Efficient Approximate Nearest Neighbor Search for Multiple Weighted $l_{p\leq 2}$ Distance Functions

Huan Hu and Jianzhong Li

Abstract—Nearest neighbor search is fundamental to a wide range of applications. Since the exact nearest neighbor search suffers from the “curse of dimensionality”, approximate approaches, such as Locality-Sensitive Hashing (LSH), are widely used to trade a little query accuracy for a much higher query efficiency. In many scenarios, it is necessary to perform nearest neighbor search under multiple weighted distance functions in high-dimensional spaces. This paper considers the important problem of supporting efficient approximate nearest neighbor search for multiple weighted distance functions in high-dimensional spaces. To the best of our knowledge, prior work can only solve the problem for the $l_2$ distance. However, numerous studies have shown that the $l_p$ distance with $p \in (0, 2)$ could be more effective than the $l_2$ distance in high-dimensional spaces. We propose a novel method, WLSH, to address the problem for the $l_p$ distance for $p \in (0, 2)$. WLSH takes the LSH approach and can theoretically guarantee both the efficiency of processing queries and the accuracy of query results while minimizing the required total number of hash tables. We conduct extensive experiments on synthetic and real data sets, and the results show that WLSH achieves high performance in terms of query efficiency, query accuracy and space consumption.

Index Terms—Approximate nearest neighbor search, multiple weighted distance functions, $l_p$ distance, locality-sensitive hashing

1 INTRODUCTION

Nearest neighbor search is fundamental to a wide variety of applications, such as recommendation, classification and pattern recognition. Let $P$ be a set of data points in a $d$-dimensional space. Given a query point $q$, the nearest neighbor search is to find a point $o^* \in P$ such that $o^*$ is the closest point to $q$.

The exact nearest neighbor search has been well studied for low-dimensional cases [1], [2], [3]. A number of methods, such as kd-tree and R-tree based methods [4], [5], have been proposed to efficiently solve the exact nearest neighbor search problem in low-dimensional spaces.

However, conventional methods become even less efficient than the brute-force linear scan when data dimensionality grows due to the “curse of dimensionality” [6]. To cope with this problem, approximate approaches are studied to trade query accuracy for query efficiency. The motivation is that approximate query results are acceptable in most scenarios. There are several approximate approaches that can perform nearest neighbor search efficiently in high-dimensional spaces [7], [8], [9], [10]. Locality-Sensitive Hashing (LSH) is a popular one among them because of its high performance in both theory and practice [10], [11]. It relies on LSH families to create hash tables and then process nearest neighbor queries over the hash tables. Essentially, an LSH family is a set of hash functions that can hash closer points into the same bucket with higher probability. A few LSH schemes have been designed based on LSH families in literature [12], [13], [14], [15], [16]. E2LSH [12], [17] and C2LSH [15] are two well-known LSH schemes among them. The former ensures that the time cost of processing a nearest neighbor query is sublinear in data size while the latter ensures that the number of hash tables required to be created grows logarithmically with data size.

While performing nearest neighbor search, distance functions are often used to compute the distance of two points. To accurately compute the “real” distance of two points in practice, weighted distance functions are usually adopted [18], [19], [20]. Briefly, a weighted distance function associates a specific weight vector with the dimensions of a multidimensional space. Thus, a distance function can potentially derive a set of weighted distance functions. In many scenarios, it is necessary to support nearest neighbor search for multiple weighted distance functions in high-dimensional spaces [20], [21], [22], [23], [24]. For example, a personalized recommender system may recommend products based on users’ preferences [21], [22]. In such case, products are represented as a set, $P$, of high-dimensional points, and users’ preferences are represented as a set, $S$, of high-dimensional weight vectors. Each weight vector in $S$ determines a weighted distance function. When a user whose preference is $\vec{W} \in S$ has already showed interest in product $o \in P$, the system would recommend the product that is the nearest neighbor of $o$ under $\vec{W}$ to the user.

Traditional LSH schemes like E2LSH and C2LSH are originally proposed to support nearest neighbor search for a distance function. They can not be directly applied to perform nearest neighbor search under multiple weighted distance functions. The fundamental reason is that two close points under a weighted distance function are possibly far away under another weighted distance function. To the best of our knowledge, SL-ALSH and S2-ALSH [25] are the only existing methods in literature that can achieve this goal. They each introduce an asymmetric LSH family on top of E2LSH, and the asymmetric LSH families allow them to process each nearest neighbor query flexibly according to

- H. Hu and J. Li are with the School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China.
E-mail: hit_huhuan@foxmail.com, lijzh@hit.edu.cn.
the weight vector attached to the query.

Unfortunately, SL-ALSH and S2-ALSH can only work for the $l_2$ distance due to the limitations of the asymmetric LSH families of them. Numerous studies over the last few decades have shown that the $l_1$ distance and the fractional distance, which is the $l_p$ distance with $0 < p < 1$, could be more effective than the $l_2$ distance in high-dimensional spaces.

Clearly, it remains a challenging problem to support efficient approximate nearest neighbor search for multiple weighted distance functions with respect to the $l_p$ distance in high-dimensional spaces for $p \in (0, 2)$.

To solve the challenging problem, we propose a novel method called WLSH in this paper. Actually, WLSH is also applicable for the $l_2$ distance. A brief overview of WLSH is as follows. First, WLSH uses the weighted LSH families that are suitable for weighted distance functions with respect to the $l_p$ distance to create hash tables, where $p \in (0, 2]$. Different weighted distance functions correspond to different weighted LSH families. Thus, multiple different groups of hash tables can be created in WLSH. Second, WLSH obtains derived weighted LSH families from the weighted LSH families to reuse hash tables. The derived weighted LSH families allow WLSH to reuse a group of hash tables to process nearest neighbor queries under different weighted distance functions. Third, WLSH partitions the given weighted distance functions into several disjoint subsets such that nearest neighbor search under each of the subsets can be supported by using a single group of hash tables. In this way, nearest neighbor search under the given weighted distance functions can be supported. The optimization goal is to process all nearest neighbor queries efficiently with accuracy guarantees while minimizing the required total number of hash tables.

Contributions. The main contributions of this paper are summarized as follows.

1. We propose weighted LSH families and derived weighted LSH families that support not only weighted $l_2$ distance functions but also weighted $l_p$ distance functions for $p \in (0, 2)$. Actually, the weighted LSH families and derived weighted LSH families can be obtained for various distance measures.

2. Based on weighted LSH families and derived weighted LSH families, a novel method, WLSH, is proposed to support efficient approximate nearest neighbor search under multiple weighted distance functions with respect to the $l_p$ distance, where $p \in (0, 2]$. WLSH offers theoretical guarantees for the query efficiency and query accuracy while minimizing the space consumption.

3. To implement the WLSH method, a preprocessing algorithm is designed to create hash tables and a search algorithm is designed to answer nearest neighbor queries over the hash tables. Assuming the number of weighted distance functions is a constant, the time and space complexities of SL-ALSH, S2-ALSH and WLSH are shown in Table 1 where $n$ is data size, $d$ is data dimensionality, and $\rho_{SL}$ and $\rho_{S2}$ are two real numbers computed by Equations 17 and 18 in Appendix A. As it can be seen, though WLSH is applicable for more than the $l_2$ distance, the space complexity of WLSH is lower than those of SL-ALSH and S2-ALSH, and the time complexity of WLSH is comparable to those of SL-ALSH and S2-ALSH when $d$ is large.

4. Reasonable trade-offs among the query efficiency, query accuracy and space consumption of WLSH are investigated in order to achieve better practical performance.

5. Extensive experiments are conducted on both synthetic and real data sets. The results show that WLSH is effective and efficient, and it performs better than SL-ALSH and S2-ALSH in many cases.

The rest of the paper is organized as follows. Section 2 provides the background knowledge and defines the problem we address. Section 3 introduces the weighted LSH families and derived weighted LSH families. The method of WLSH is presented in Section 4. The experimental results are reported in Section 5. We review related work in Section 6 and conclude the paper in Section 7.

## 2 Preliminaries

Sections 2.1, 2.2 and 2.3 review the LSH approach in the context of approximate nearest neighbor search under a distance function. Section 2.4 defines our problem.

### 2.1 $c$-NN Search and $(R, c)$-NN Search

Assume $\mathcal{X}$ is a $d$-dimensional space and $D(\cdot, \cdot)$ is a distance function defined on $\mathcal{X}$. Let $B(x, R) = \{ y \mid D(x, y) \leq R \}$. In a typical LSH scheme, approximate nearest neighbor search is formalized as the $c$-approximate nearest neighbor ($c$-NN for short) search problem as follows.

**Definition 1.** ($c$-NN Search [11]). Given a $d$-dimensional space $\mathcal{X}$, a distance function $D(\cdot, \cdot)$, an approximation ratio $c > 1$, a set of data points $P \subset \mathcal{X}$ and a user-specified query point $q \in \mathcal{X}$, the $c$-NN search is to return a point $o \in P$ such that $D(q, o) \leq c \times D(q, o^*)$, where $o^*$ is the exact nearest neighbor of $q$ in $P$.

As will be shown in Section 2.3 an LSH scheme solves the $c$-NN search problem by reducing it to the $c$-approximate $R$-near neighbor ($(R, c)$-NN for short) search problem defined as follows.

**Definition 2.** ($(R, c)$-NN Search [11]). Given a $d$-dimensional space $\mathcal{X}$, a distance function $D(\cdot, \cdot)$, a search radius $R > 0$, an approximation ratio $c > 1$, a set of data points $P \subset \mathcal{X}$ and a user-specified query point $q \in \mathcal{X}$, the $(R, c)$-NN search is to return a point in $B(q, cR) \cap P$ if $B(q, R) \cap P \neq \emptyset$.

Essentially, the $(R, c)$-NN search problem is a decision version of the $c$-NN search problem [15]. For convenience, the result of the $c$-NN search is called a $c$-NN, and the result of the $(R, c)$-NN search is called a $(R, c)$-NN.
2.2 LSH Family

Formally, an LSH family is defined as follows.

**Definition 3. (LSH FAMILY)** [11]. An LSH family \( \mathcal{H} = \{ h : \mathcal{X} \rightarrow U \} \) is called \((R,cR,P_1,P_2)\)-sensitive if for any \( x,y \in \mathcal{X} \)

1. if \( y \in B(x,R) \), then \( \Pr[h(x) = h(y)] \geq P_1 \);
2. if \( y \notin B(x,cR) \), then \( \Pr[h(x) = h(y)] \leq P_2 \).

In order for an LSH family to be useful, \( P_1 \) has to be greater than \( P_2 \). Usually, a collision probability function \( P(\cdot) \) can be obtained for an LSH family such that \( P_1 = P(R) \) and \( P_2 = P(cR) \).

We focus on the \( l_p \)-distance \((p \in (0,2))\) in this paper. The distance function with respect to the \( l_p \)-distance is

\[
D(x, y) = \sqrt[p]{\sum_{i=1}^{d} |x_i - y_i|^p}
\]

where \( x = (x_1, x_2, \ldots, x_d) \in \mathcal{X} = \mathbb{R}^d \) and \( y = (y_1, y_2, \ldots, y_d) \in \mathcal{X} = \mathbb{R}^d \). A classical LSH family for the \( l_p \)-distance is \( \mathcal{H}_{\bar{a},b} = \{ h_{\bar{a},b} : \mathcal{X} \rightarrow U \} \), where

\[
h_{\bar{a},b}(x) = \left\lfloor \frac{\bar{a} \cdot x + b}{w} \right\rfloor,
\]

\( \bar{a} \) is a \( d \)-dimensional vector where each entry is chosen independently from the \( p \)-stable distribution [17], [30] (e.g., the 2-stable distribution is the standard normal distribution), \( w \) is a user-specified constant, and \( b \) is a real number chosen uniformly at random from \([0, w]\). The collision probability function is \( P_p(x) = \int_0^x \int_{x}^{\infty} f_{\mathcal{F}_p}(z)dzdw \), where \( f_{\mathcal{F}_p}(\cdot) \) is the PDF of the absolute value of the \( p \)-stable distribution [17].

2.3 E2LSH and C2LSH

In the following, we review two well-known LSH schemes, E2LSH [12], [31] and C2LSH [15]. The preprocessing algorithms and search algorithms for the \((R,c)\)-NN search and \(c\)-NN search are presented for each scheme.

2.3.1 E2LSH

We focus on the \((R,c)\)-NN search first and then reduce the \(c\)-NN search to the \((R,c)\)-NN search.

**\((R,c)\)-NN Search.** The preprocessing algorithm and search algorithm for the \((R,c)\)-NN search are as follows.

**Preprocessing algorithm.** Given a \( \mathcal{H},P_1,P_2 \)-sensitive LSH family \( \mathcal{H} \), a compound hash function is of the form: \( g = (h_1,h_2,\ldots,h_m) \), where \( h_1, h_2, \ldots, h_m \) are LSH functions chosen independently and uniformly at random from \( \mathcal{H} \). The preprocessing algorithm first generates \( L \) compound hash functions \( g_1, g_2, \ldots, g_L \) independently. Then, it creates \( L \) hash tables according to the compound hash functions. As a result, each data point \( o \in P \) is hashed into bucket \( g_i(o) \) of the \( i \)-th hash table for \( 1 \leq i \leq L \).

**Search algorithm.** Given a query point \( q \in \mathcal{X} \), the search algorithm takes the data points in buckets \( g_1(q), g_2(q), \ldots, g_L(q) \) as candidates and computes their distances to \( q \) to check if they are \((R,c)\)-NNs of \( q \). It stops immediately after a \((R,c)\)-NN of \( q \) is found or \( tL \) candidates are checked, where \( t \) is a constant.

1. In this paper, \( \bar{x} \) denotes the vector representation of \( x \in \mathcal{X} \).

Parameter settings. Let \( n \) be the number of points in \( P \). When we set \( m = \log_1/P_1 \), \( L = n^p \) where \( \rho = \log_1/P_2 \), \( \log_1/P_1 < 1 \), the search algorithm above can find a \((R,c)\)-NN of \( q \) in \( P \) with a constant probability if \( B(q,R) \cap P \neq \emptyset \).

**c-NN Search.** Let \( r_{\min} \) and \( r_{\max} \) be the smallest and the largest possible distances between two points respectively. The \( c\)-NN search can be performed by issuing a series of \((R,c)\)-NN search with increasing radii \( R = r_{\min}, c_{\min}r_{\min}, \ldots, c_{\log_1/r_{\max}/r_{\min}}r_{\min} \). Thus, the preprocessing algorithm for the \( c\)-NN search simply calls the preprocessing algorithm for the \((R,c)\)-NN search at radii \( R = r_{\min}, c_{\min}r_{\min}, \ldots, c_{\log_1/r_{\max}/r_{\min}}r_{\min} \) respectively. Accordingly, the search algorithm for the \( c\)-NN search simply calls the search algorithm for the \((R,c)\)-NN search at radii \( R = r_{\min}, c_{\min}r_{\min}, \ldots, c_{\log_1/r_{\max}/r_{\min}}r_{\min} \) respectively, and it stops immediately after a \((R,c)\)-NN is found or \( tL \) candidates are checked at some search radius.

2.3.2 C2LSH

The scheme C2LSH is suitable for the \( l_p \)-distance. It relies on two LSH families for the \( l_p \)-distance, \( \mathcal{H}_{\bar{a},b} = \{ h_{\bar{a},b} : \mathcal{X} \rightarrow U \} \) and \( \mathcal{H}_{\bar{a},b}^l = \{ h_{\bar{a},b}^l : \mathcal{X} \rightarrow U \} \), where

\[
h_{\bar{a},b}(x) = \left\lfloor \frac{\bar{a} \cdot x + b}{w} \right\rfloor, \quad h_{\bar{a},b}^l(x) = \left\lfloor \frac{h_{\bar{a},b}(x)}{l} \right\rfloor.
\]

\( \bar{a} \) and \( w \) are the same as in Equation 2 and \( b^* \) is a random number chosen uniformly at random from \([0, c_{\log_1/r_{\max}/r_{\min}}] \) [14]. Lemma 1 in this paper indicates that \( \mathcal{H}_{\bar{a},b}^l \) is \((x,y,P_1(x),P_2(y))\)-sensitive and \( \mathcal{H}_{\bar{a},b} \) is \((x,y,P_1(x),P_2(y))\)-sensitive for any \( 0 < x < y \) and \( l \in \{c,c^2,\ldots,c_{\log_1/r_{\max}/r_{\min}}\} \).

Buckets defined by an LSH function from \( \mathcal{H}_{\bar{a},b}^l \) are called level-1 buckets, and buckets defined by an LSH function from \( \mathcal{H}_{\bar{a},b} \) are called level-\( l \) buckets. From Equation 3 it is easy to see that a level-1 bucket \( x \) consists of \( l \) consecutive level-1 buckets \( x \times l, x \times l+1, \ldots, x \times l+l-1 \). Therefore, the exploration of a level-1 bucket can be implemented by probing the corresponding \( l \) level-1 buckets. The technique is called virtual rehashing.

**\((R,c)\)-NN Search.** Assume the search radius is \( R = r_{\min,l} \), where \( l \in \{1,c,c^2,\ldots,c_{\log_1/r_{\max}/r_{\min}}\} \). The preprocessing algorithm and search algorithm for the \((R,c)\)-NN search are described as follows.

**Preprocessing algorithm.** The algorithm first chooses \( \beta \) LSH functions \( h_{\bar{a},b,1}, h_{\bar{a},b,2}, \ldots, h_{\bar{a},b,\beta} \) independently and uniformly at random from \( \mathcal{H}_{\bar{a},b} \). Then, it creates \( \beta \) hash tables according to the LSH functions. As a result, each data point \( o \in P \) is hashed into bucket \( h_{\bar{a},b,i}(o) \) of the \( i \)-th hash table for \( 1 \leq i \leq \beta \).

**Search algorithm.** Some concepts need to be introduced first. Given a query point \( q \in \mathcal{X} \), the collision number of a data point \( o \in P \) is the number of buckets where \( o \) collides with \( q \). A data point \( o \in P \) is called frequent if its collision number is greater than or equal to a collision threshold \( \mu \). \( \gamma \) is the allowable percentage of false positives which are frequent points but not \((R,c)\)-NNs.

Let \( h_{\bar{a},b,i}^l(q) \) be the level-\( l \) bucket with respect to \( h_{\bar{a},b,i} \) and a query point \( q \in \mathcal{X} \) for \( 1 \leq i \leq \beta \). The search algorithm takes the frequent points in buckets
Different radii for the indexing technique allows C2LSH not to create hash tables at different radii. From [15], we have Pr[|P1| holds] \geq 1 - \epsilon and Pr[|P2| holds] > 1/2 by setting $\beta$ and $\mu$ to:

$$\beta = \left[ \frac{\ln \frac{1}{\epsilon}}{2(P_{t_p}(x) - P_{t_p}(y))} (1 + z)^2 \right]$$

$$\mu = \frac{zP_{t_p}(x) + P_{t_p}(y)}{1 + z} \beta$$

where $z = \sqrt{\ln \frac{2}{\epsilon}}$, $x = r_{min}$ and $y = cr_{min}$. As a result, a $(R, c)$-NN of $q$ in $P$ can be found with a constant probability if $B(q, R) \cap P \neq \emptyset$. In practice, $\epsilon$ and $\gamma$ can be set to 0.01 and 100/n respectively [15].

**c-NN Search.** Different from E2LSH, the virtual rehashing technique allows C2LSH not to create hash tables at different radii for the c-NN search. Thus, the preprocessing algorithm for the c-NN search is the same as the preprocessing algorithm for the $(R, c)$-NN search. The search algorithm for the c-NN search calls the search algorithm for the $(R, c)$-NN search at radii $R = r_{min}, cr_{min}, \ldots, c^{\lceil \log_{c} r_{max}/r_{min} \rceil} r_{min}$ respectively, and it stops immediately after a $(R, c)$-NN is found or $\gamma n$ candidates are checked at some search radius.

### 2.4 Problem Description

Generally, for any distance measure, the distance function is of the form $d(x, y) = f(x_1, y_1, x_2, y_2, \ldots, x_d, y_d)$, where $f$ is a function depending on the distance measure, $\vec{x} = (x_1, x_2, \ldots, x_d) \in X$ and $\vec{y} = (y_1, y_2, \ldots, y_d) \in X$. Based on this, a weighted distance function is defined as follows.

**Definition 4. (Weighted Distance Function).** Given a distance function $D(x, y) = f(x_1, y_1, x_2, y_2, \ldots, x_d, y_d)$ and a weight vector $\vec{w} = (w_1, w_2, \ldots, w_d)$ ($w_i > 0$ for $1 \leq i \leq d$), the weighted distance function with respect to them is $D_{\vec{w}}(x, y) = f(w_1x_1, w_1y_1, w_2x_2, w_2y_2, \ldots, w_dx_d, w_dy_d)$.

Please note that weighted distance functions can take different forms as in [23, 33, 34]. Actually, these forms are equivalent to the one in Definition 4. It can be seen from Definition 4 that a weighted distance function is identified by the associated weight vector for a given distance measure. Thus, we sometimes refer to a weighted distance function by the associated weight vector for ease of discussion.

A weighted distance function can be viewed as a special distance function. Thus, the LSH approach introduced in Sections 2.1, 2.2 and 2.3 can be directly extended to support approximate nearest neighbor search for a weighted distance function. Let $B_{\vec{w}}(x, R) = \{y \mid D_{\vec{w}}(x, y) \leq R\}$. Then the c-NN search problem is extended to the weighted c-approximate nearest neighbor (c-WNN for short) search problem in Definition 5 and the $(R, c)$-NN search problem is extended to the weighted c-approximate R-near neighbor $(R, c)$-WNN for short) search problem in Definition 6. As it can be seen, the c-WNN search problem is a formalization of approximate nearest neighbor search under a weighted distance function. Given a proper LSH family, an LSH scheme, e.g., E2LSH or C2LSH, solves the c-WNN search problem by reducing it to the $(R, c)$-WNN search problem.

**Definition 5. (c-WNN Search).** Given a $d$-dimensional space $X$, a distance function $D(\cdot, \cdot)$, a $d$-dimensional weight vector $\vec{w}$, an approximation ratio $c > 1$, a set of data points $P \subset X$ and a user-specified query point $q \in X$, the c-WNN search is to return a point $o \in P$ such that $D_{\vec{w}}(q, o) \leq c \times D_{\vec{w}}(q, o^*)$, where $o^*$ is the exact nearest neighbor of $q$ in $P$ under $\vec{w}$.

**Definition 6. ((R, c)-WNN Search).** Given a $d$-dimensional space $X$, a distance function $D(\cdot, \cdot)$, a $d$-dimensional weight vector $\vec{w}$, a search radius $R > 0$, an approximation ratio $c > 1$, a set of data points $P \subset X$ and a user-specified query weight vector $\vec{w}_i \in S$, and a user-specified query point $q \in X$.

#### Problem Statement

**Input:** A $d$-dimensional space $X$, a distance function $D(\cdot, \cdot)$, a set of $d$-dimensional weight vectors $S$, an approximation ratio $c > 1$, a result cardinality $k \geq 1$, a set of data points $P \subset X$, and a user-specified query weight vector $\vec{w}_i \in S$, and a user-specified query point $q \in X$.

**Output:** kANNSs = $\{o_1, o_2, \ldots, o_k\}$ $\subset P$ that satisfies $D_{\vec{w}_i}(q, o_j) \leq c \times D_{\vec{w}_i}(q, o^*_j)$ for $1 \leq j \leq k$, where the points $o^*_1, o^*_2, \ldots, o^*_k$ are the exact $k$-nearest neighbors of $q$ in $P$ under $\vec{w}_i$.

If $|S|$ is the cardinality of the input weight vector set $S$, is equal to 1, then the problem is called the weighted c-approximate k-nearest neighbor ((c, k)-WNN for short) search problem. The (c, k)-WNN search problem is a formalization of approximate k-nearest neighbor search under a weighted distance function. It can be solved by repeating the preprocessing algorithm for the c-WNN search and modifying the search algorithm for the c-WNN search to stop it after $k$ ($R, c$)-WNNs are found or $k$ more candidates than allowed are checked at some search radius [15].

It is obvious that our problem is to support the (c, k)-WNN search for each weight vector in $S$. A naive method for solving our problem is applying the above solution to the (c, k)-WNN search problem for each weight vector in $S$. As a result of the naive method, the required total number of hash tables is linear in $|S|$. When $|S|$ is not small, the space overhead could be hard to tolerate in practice. Therefore, we would like to propose a new method which is more space-efficient than the naive method while ensuring that the time efficiency is still high. We concentrate on the $l_p$ distance.

For convenience, the (c, k)-WNN search under a weight vector $\vec{w}_i$ is defined as the (c, k)-WNN search where the weighted distance function used is $D_{\vec{w}_i}(\cdot, \cdot)$, and the (c, k)-WNN search under a weight vector set $S$ is defined as the (c, k)-WNN search where the weighted distance function used is in the range $\{D_{\vec{w}_i}(\cdot, \cdot) \mid \vec{w}_i \in S\}$. The result of the (c, k)-WNN search under a weight vector $\vec{w}_i$ is called
a \((c, k)\)-WNN under \(\tilde{W}\). Similar definitions are also introduced for the \((R, c)\)-WNN search and the \(c\)-WNN search.

Since the techniques for the \(c\)-WNN search can almost cover the techniques for the \((c, k)\)-WNN search, we will limit ourselves to the \(c\)-WNN search before introducing the method of WLSH. The frequently-used notations in the paper are summarized in Table 2.

| Notation | Description |
|----------|-------------|
| \(\mathcal{X}\) | A \(d\)-dimensional space |
| \(P\) | A set of data points in \(\mathcal{X}\) |
| \(n\) | The cardinality of \(P\) |
| \(q\) | A query point |
| \(\tilde{W}\) | A weight vector |
| \(D_{\tilde{W}}(\cdot, \cdot)\) | The weighted distance function identified by \(\tilde{W}\) |
| \(B_{\tilde{W}}(x, R)\) | The ball centered at \(x\) with radius \(R\) under \(\tilde{W}\) |
| \(r_{\max}(\tilde{w}_{\min})\) | The largest (smallest) possible distance between two points under \(\tilde{W}\) |
| \(\mu_{\tilde{W}}\) | The collision threshold for \(\tilde{W}\) |
| \(\gamma\) | The allowable percentage of false positives |
| \(\beta_{\tilde{W}}\) | The required number of hash tables for \(\tilde{W}\) |
| \(S\) | A weight vector set |
| \(c\) | Approximation ratio |
| PDF (PMF) | Abbreviation of probability density (mass) function |

3 LSH FAMILIES FOR WEIGHTED DISTANCE FUNCTIONS

In this section, we introduce two kinds of LSH families for weighted distance functions, namely, weighted LSH families and derived weighted LSH families.

3.1 Weighted LSH Family

Traditional LSH families are designed to hash closer points under distance functions into the same buckets with higher probabilities, they are not suitable for weighted distance functions. We propose weighted LSH families to cope with this problem. Essentially, weighted LSH families generalize traditional LSH families to weighted distance functions.

Given a weight vector \(\tilde{W}\), we have \(D_{\tilde{W}}(o, q) = D(\tilde{W} \circ o, \tilde{W} \circ q)\) from Definition 4 where \(o \in P\), \(q \in \mathcal{X}\), and \(\circ\) is an element-wise product. This suggests that searching for a \(c\)-WNN of \(q \in P\) under \(\tilde{W}\) is equivalent to searching for a \(c\)-NN of \(\tilde{W} \circ q \in \tilde{W} \circ P\). Therefore, given a traditional LSH family \(H = \{h : \mathcal{X} \rightarrow U\}\), we can directly obtain a weighted LSH family \(H_{\tilde{W}} = \{h : \tilde{W} \circ \mathcal{X} \rightarrow U\}\), where \(\tilde{W} \circ \mathcal{X} = \{\tilde{W} \circ x | x \in \mathcal{X}\}\). For convenience, define \(h_{\tilde{W}}(x) = h(\tilde{W} \circ x)\) for \(x \in \mathcal{X}\) such that \(H_{\tilde{W}} = \{h_{\tilde{W}} : \mathcal{X} \rightarrow U\}\).

Next, we determine \(P_{1,\tilde{W}}\) and \(P_{2,\tilde{W}}\), such that \(H_{\tilde{W}}\) is \((R, cR, P_{1,\tilde{W}}, P_{2,\tilde{W}})\)-sensitive. It is easy to see that \(H_{\tilde{W}}\) involves the same set of random variables as \(H\). Thus, the first step is to set \(P_{1,\tilde{W}} = P_{\tilde{W}}(R)\) and \(P_{2,\tilde{W}} = P_{\tilde{W}}(cR)\), which is purely a mathematical problem.

The weighted LSH family obtained from \(H_{\tilde{W}}\) in Section 2.2 is \(H_{\tilde{a},\tilde{b},\tilde{W}} = \{h_{\tilde{a},\tilde{b},\tilde{W}} : \mathcal{X} \rightarrow U\}\), where

\[
h_{\tilde{a},\tilde{b},\tilde{W}}(x) = \frac{\tilde{a} \cdot (\tilde{W} \circ x) + b}{w},
\]

and the PDFs of \(\tilde{a}\) and \(b\) are the same as for \(H_{\tilde{a},\tilde{b}}\). We can easily know from [7] that the collision probability function for \(H_{\tilde{a},\tilde{b},\tilde{W}}\) is

\[
P_{\tilde{a},\tilde{b},\tilde{W}}(r) = P_{\tilde{a},\tilde{b}}(r) = \int_0^r F_{\tilde{W}}(x)(1 - \frac{r}{x}) \, dx.
\]

3.2 Derived Weighted LSH Family

Given a weighted LSH family \(H_{\tilde{W}} = \{h_{\tilde{W}} : \mathcal{X} \rightarrow U\}\) and a weight vector \(\tilde{W}'\), the derived weighted LSH family with respect to them is denoted by \(H_{\tilde{W} \rightarrow \tilde{W}'} = \{h_{\tilde{W} \rightarrow \tilde{W}'} : \mathcal{X} \rightarrow U\}\). Suppose a group of hash tables have been created according to weighted LSH functions from \(H_{\tilde{W}}\) to support the \(c\)-WNN search under \(\tilde{W}\). The goal of \(H_{\tilde{W} \rightarrow \tilde{W}'}\) is to reuse the hash tables for the \(c\)-WNN search under \(\tilde{W}'\) (assume \(\tilde{W} \neq \tilde{W}'\)). First, we let \(h_{\tilde{W} \rightarrow \tilde{W}'}(x) = h_{\tilde{W}}(x)\) for \(x \in \mathcal{X}\). That is, \(H_{\tilde{W} \rightarrow \tilde{W}'}\) hashes points in the same way as \(H_{\tilde{W}}\). The next step is to determine \(P_{1,\tilde{W} \rightarrow \tilde{W}'}\) and \(P_{2,\tilde{W} \rightarrow \tilde{W}'}\), such that \(H_{\tilde{W} \rightarrow \tilde{W}'}\) is \((R, cR, P_{1,\tilde{W} \rightarrow \tilde{W}'}(R), P_{2,\tilde{W} \rightarrow \tilde{W}'}(R))\)-sensitive. If \(P_{1,\tilde{W} \rightarrow \tilde{W}'}(R) > P_{2,\tilde{W} \rightarrow \tilde{W}'}(R)\), then \(H_{\tilde{W} \rightarrow \tilde{W}'}\) is an effective LSH family under \(\tilde{W}'\). In order to determine \(P_{1,\tilde{W} \rightarrow \tilde{W}'}\) and \(P_{2,\tilde{W} \rightarrow \tilde{W}'}\), we make the following assumption.

**Assumption 1.** The collision probability \(P_{\tilde{W}}(r)\) is inversely proportional to \(r\).

In fact, the assumption is reasonable and it holds for all the weighted LSH families in the paper. According to Definition 3, \(H_{\tilde{W} \rightarrow \tilde{W}'}\) is \((R, cR, P_{1,\tilde{W} \rightarrow \tilde{W}'}(R), P_{2,\tilde{W} \rightarrow \tilde{W}'}(R))\)-sensitive if and only if (1) any point in \(B_{\tilde{W}}(q, R)\) collides with \(q\) with probability at least \(P_{1,\tilde{W} \rightarrow \tilde{W}'}(R)\), and (2) any point not in \(B_{\tilde{W}}(q, cR)\) collides with \(q\) with probability at most \(P_{2,\tilde{W} \rightarrow \tilde{W}'}(R)\). The two conditions are formulated as:

\[
P_{1,\tilde{W} \rightarrow \tilde{W}'} \leq \min_{o \in B_{\tilde{W}'}(q, R)} \Pr[h_{\tilde{W} \rightarrow \tilde{W}'}(o) = h_{\tilde{W} \rightarrow \tilde{W}'}(q)]
\]

\[
= \min_{o \in B_{\tilde{W}'}(q, R)} \Pr[h_{\tilde{W}}(o) = h_{\tilde{W}}(q)]
\]

\[
P_{2,\tilde{W} \rightarrow \tilde{W}'} \geq \max_{o \in B_{\tilde{W}'}(q, cR)} \Pr[h_{\tilde{W} \rightarrow \tilde{W}'}(o) = h_{\tilde{W} \rightarrow \tilde{W}'}(q)]
\]

\[
= \max_{o \in B_{\tilde{W}'}(q, cR)} \Pr[h_{\tilde{W}}(o) = h_{\tilde{W}}(q)]
\]
Define $R^\dagger$ as an upper bound of the largest possible distance between $q$ and $o \in B_{\vec{W}^s}(q, R)$ under $\vec{W}$, and define $(cR)^\dagger$ as a lower bound of the smallest possible distance between $q$ and $o \notin B_{\vec{W}^s}(q, cR)$ under $\vec{W}$. That is,

$$R^\dagger \geq \max_{o \in B_{\vec{W}^s}(q, R)} D_{\vec{W}}(o, q)$$

$$(cR)^\dagger \leq \min_{o \notin B_{\vec{W}^s}(q, cR)} D_{\vec{W}}(o, q)$$

Combining Equation 9 and Assumption I, we obtain:

$$P_{\vec{W}}(R^\dagger) \leq \min_{o \in B_{\vec{W}^s}(q, R)} \Pr[h_{\vec{W}^s}(o) = h_{\vec{W}}(q)]$$

$$P_{\vec{W}}((cR)^\dagger) \geq \max_{o \notin B_{\vec{W}^s}(q, cR)} \Pr[h_{\vec{W}^s}(o) = h_{\vec{W}}(q)]$$

Further, combining Equations 8 and 10, we can safely set $P_{\vec{W}}(R^\dagger) = P_{\vec{W}}(R^\dagger)$ and $P_{\vec{W}}((cR)^\dagger)$. As a result, $H_{\vec{W}^s}$ is $\langle R, cR, P_{\vec{W}}(R^\dagger), P_{\vec{W}}((cR)^\dagger) \rangle$-sensitive.

In order for $H_{\vec{W}^s}$ to be useful, $R^\dagger < (cR)^\dagger$ should hold so that $P_{\vec{W}}(R^\dagger) > P_{\vec{W}}((cR)^\dagger)$. Actually, $R^\dagger < (cR)^\dagger$ may not hold even when $R^\dagger$ and $(cR)^\dagger$ are both tight bounds. In the following Theorem 1, we give $R^\dagger$'s and $(cR)^\dagger$'s for the $l_p$ distance, Hamming distance and angular distance. The proof can be found in Appendix C.

**Theorem 1.** Given any $R > 0$ and any $H_{\vec{W}^s} - \vec{W}$, where $\vec{W} = (w_1, w_2, \ldots, w_d)$ and $\vec{W}' = (w_1', w_2', \ldots, w_d')$, we have:

1. For the $l_p$ distance, $R^\dagger = R \max_{1 \leq i \leq d}(w_i/w_i')$ and $(cR)^\dagger = cR \min_{1 \leq i \leq d}(w_i/w_i')$.
2. For the Hamming distance, $R^\dagger = R \max_{1 \leq i \leq d}(w_i/w_i')$ and $(cR)^\dagger = cR \min_{1 \leq i \leq d}(w_i/w_i')$.
3. For the angular distance, $R^\dagger = \arccos(\max\{\pm 1, Y\})$ and $(cR)^\dagger = \arccos(\min(1, Y\sqrt{Y}))$, where $Y = \cos(cR)/(N - M)/M' = Y = \cos(cR)/M' + (M - N)/N'M = \max_{1 \leq i \leq d}(w_i^2/w_i')^2$, and $N = \min_{1 \leq i \leq d}(w_i^2/w_i')$.

**4 WLSH**

Our problem is to efficiently support the $(c, k)$-WNN search under a given weight vector set $S$ for the $l_p$ distance. Recall the overview of WLSH in Section 3. WLSH addresses the problem by partitioning $S$ into several disjoint subsets and then supporting the $(c, k)$-WNN search under each of the subsets respectively. The partition method can ensure that the $(c, k)$-WNN search under each obtained subset of $S$ requires only a single group of hash tables. In Section 4.1 we introduce how WLSH supports the $(c, k)$-WNN search under a subset of $S$ by creating a group of hash tables, and in Section 4.2 we introduce how WLSH partitions $S$ into several disjoint subsets appropriately. Finally, we present a preprocessing algorithm and a search algorithm for the $(c, k)$-WNN search under $S$.

4.1 $(c, k)$-WNN Search under a Subset of $S$

We base WLSH on the LSH scheme of C2LSH [15]. Thus, the weighted LSH families given by Equation 2 are used for WLSH. We focus on the $c$-WNN search first. Let $S^o$ be a subset of $S$. Suppose the $c$-WNN search under $S^o$ will be supported using a group of hash tables created according to weighted LSH functions from $H_{\vec{W}^s}$, where $\vec{W}^s = \{w_1^s, w_2^s, \ldots, w_d^s\}$ is some weight vector in $S$. We need to determine the required number of hash tables $\beta_{\vec{W}^s}$ and the collision threshold $\mu_{\vec{W}^s}$ for each $\vec{W}^s \in S^o$ from Section 2.3.2. The number of hash tables required to be created is $\beta_{\vec{W}^s} = \max_{\vec{W} \in S^o}(\beta_{\vec{W}})$. To obtain the parameters $B_{\vec{W}^s} = \{(\beta_{\vec{W}^s}, \mu_{\vec{W}^s}) \mid \vec{W} \in S^o\} \cup \{\beta_{S^o}\}$, the following Lemma 1 is necessary.

**Lemma 1.** Given any two real numbers $x$ and $y$ satisfying $0 < x < y$, we have:

1. $H_{\vec{W}^s}$ is $(x, y, P_{\vec{W}^s}(x), P_{\vec{W}^s}(y))$-sensitive.
2. $H_{\vec{W}^s}$ is $(x, y, P_{\vec{W}^s}(x), P_{\vec{W}^s}(y))$-sensitive for $l \in \{c, c^2, \ldots, c^{\log_c w_{\text{max}}/w_{\text{min}}^o}\}$.

Let $H_{\vec{W}^s}$ be the derived weighted LSH family with respect to $H_{\vec{W}^s}$ and $\vec{W}^s$, for $\vec{W}^s \in S^o$, and let $H_{\vec{W}^s}$ be the derived weighted LSH family with respect to $H_{\vec{W}^s}$ and $\vec{W}^s$. Based on Theorem 1 and Lemma 1, we can obtain the following Lemma 2.

**Theorem 2.** Given any $\vec{W} = \{w_1, w_2, \ldots, w_d\} \in S^o$ and any two real numbers $x$ and $y$ satisfying $0 < x^\dagger < y^\dagger$, we have:

1. $H_{\vec{W}^s}$ is $(x, y, P_{\vec{W}^s}(x), P_{\vec{W}^s}(y))$-sensitive.
2. $H_{\vec{W}^s}$ is $(x, y, P_{\vec{W}^s}(x), P_{\vec{W}^s}(y))$-sensitive for $l \in \{c, c^2, \ldots, c^{\log_c w_{\text{max}}/w_{\text{min}}^o}\}$.

The proofs of Lemma 1 and Theorem 2 are shown in Appendices D and E respectively. To guarantee the correctness of Lemma 1 and Theorem 2, the PDF of $b^\dagger$ involved should be set to $b^\dagger = 1/\log_{c^r} r_{\text{max}}/r_{\text{min}}$, where $r_{\text{max}}/r_{\text{min}}$ is a real number greater than or equal to $\max_{\vec{W} \in S^o}(r_{\vec{W}^s}/r_{\vec{W}^s})$. In the experiments in Section 5, $w$ is empirically set to $r_{\vec{W}^s}/r_{\vec{W}^s}$ to achieve good performance.

The $c$-WNN search under $\vec{W}^s \in S^o$ can be performed by issuing a series of $(R, c)$-WNN search under $\vec{W}^s$ with increasing radii $R = r_{\vec{W}^s}/r_{\vec{W}^s}, \ldots, r_{\vec{W}^s}/r_{\vec{W}^s}$, $r_{\vec{W}^s}/r_{\vec{W}^s}$, following C2LSH, a $(R, c)$-WNN of $q$ in $P_{\vec{W}^s}$ can be found if the following two properties hold:

$P_{\vec{W}^s}$: If $o \in B_{\vec{W}^s}(q, R) \cap P$, then $o$ is frequent, that is, the collision number of $o$ is at least $\mu_{\vec{W}^s}$.

$P_{\vec{W}^s}$: The total number of false positives which are frequent points but not $(R, c)$-WNNs is less than $\gamma n$.

Combining Theorem 2 and Equations 4 and 5, we can easily know that $P_{\vec{W}^s}$ holds $\geq 1 - \epsilon$ and $P_{\vec{W}^s}$ holds $\geq 1/2$ if $\beta_{\vec{W}}$ and $\mu_{\vec{W}}$ are set to:

$$\beta_{\vec{W}} = \frac{\ln 1}{2(P_{\vec{W}^s}(x^\dagger) - P_{\vec{W}^s}(y^\dagger))^2} (1 + z)^2$$

$$\mu_{\vec{W}} = \frac{z P_{\vec{W}^s}(x^\dagger) + P_{\vec{W}^s}(y^\dagger)}{1 + z}$$

where $z = \sqrt{\ln 1/\epsilon^2} x = r_{\vec{W}^s}/r_{\vec{W}^s}$, $y = c r_{\vec{W}^s}$ and $\vec{W}^s \in S^o$ As in [15], we set $\epsilon = 0.01$ and $\gamma = 100/n$. Therefore, $\beta_{\vec{W}} = O(\ln n)$ for $\vec{W}^s \in S^o$. Please note that the partition method
introduced in Section 4.2 will guarantee that $x^t < y^t$ holds for $x = r_{\min}^W$ and $y = c_{\max}^W$ so that Theorem 2 is available to derive Equations [11] and [12].

Now we give the preprocessing algorithm and search algorithm for the $(c, k)$-WNN search under $S^\circ$.

The preprocessing algorithm is straightforward and as follows. First, Function Compu

We introduce the partition method first. Suppose that $S$ will be partitioned into the set of disjoint subsets $\mathcal{S}$ and the $(c, k)$-WNN search under $S_1 \in \mathcal{S}$ will be supported using a group of hash tables created according to weighted LSH functions from $\mathcal{H}_{\bar{a}, \bar{b}, r}$. Let $S^* = \{W^*_S, S \in \mathcal{S}\}$ and $W^*_S$ is some weight vector in $S$. Let $S^+ = \{W^*_S, S \in \mathcal{S}\}$. From Section 4.1 we need to determine the parameters $B_{S^+} = \{(\beta_{W^*_S}, \mu_{W^*_S}) | W^*_S \in S\} \cup \{\beta_{S}\}$ for $S \in \mathcal{S}$. Let $B_{S} = \{B_{S} | S \in \mathcal{S}\}$. The total number of hash tables required to be created is $\beta_S = \sum_{S \in \mathcal{S}} \beta_S$.

In order to efficiently perform the $(c, k)$-WNN search under $S$, we use a pre-specified threshold $\tau$ to limit the required number of hash tables for each weight vector in $S$. Moreover, it is desired to minimize $\beta_S$. As a result, the problem of partitioning $S$ is formulated as the below optimization problem:

$$\min_{\beta_S} \beta_S \quad \text{s.t.} \quad \sum_{S \in \mathcal{S}} \beta_S = \beta_S \leq \tau, S \in \mathcal{S} \land i \neq j,$$

The threshold $\tau$ should be set to ensure the existence of a solution of the above optimization problem. Consider an extreme case: the $(c, k)$-WNN search under each weight vector in $S$ is supported using an individual group of hash tables. Denote $\max_{W^*_S \in S^+} (\beta_{W^*_S})$ in this case by $\tau_{\min}$. Then $\tau$ can be set to a value no less than $\tau_{\min}$.

We determine $S^*$, $\mathcal{S}$ and $B_{S}$ by solving the above optimization problem. The steps are shown in Function $\text{Partition}(\cdot)$ and explained as follows.

\textit{Step 1} (Lines 1-9). For each $W_i \in S$, suppose $H_{\bar{a}, \bar{b}, r} W_i$ will be used for creating hash tables and then construct all subsets of $S$ that satisfy the two conditions: (1) The required number of hash tables for the $(c, k)$-WNN search under each of the subsets is less than or equal to $\tau$; (2) No more weight vectors in $S$ can be added to any of the subsets without increasing the required number of hash tables for the $(c, k)$-WNN search under it.

\textit{Step 2} (Lines 10-11). Take each of the subsets of $S$ obtained in Step 1 as a weighted set with the weight being the required number of hash tables for the $(c, k)$-WNN search under it. Further, take all these weighted sets, denoted by $Y$, as the input of the weighted set cover problem. The output will be a weighted set cover $\bar{Y} \subseteq Y$ that covers $S$ and has a minimal sum of weights. Actually, the reason why we use Condition (2) in Step 1 is that it can dramatically reduce the input size while not increasing the sum of weights of the output weighted set cover. As it can be seen, the minimum required total number of hash tables for the $(c, k)$-WNN search under $S$ is equal to the sum of weights of $\bar{Y}$. Once $\bar{Y}$ is produced, $S^*$ can be determined immediately.

```plaintext
Function ComputeParam(S^o, W^o, c)
1. foreach W_i \in S^o do
2.   Compute \beta_{W_i} and \mu_{W_i} according to Equations [11] and [12]
3.   \beta_S = \max_{W_i \in S^o} (\beta_{W_i})
4.   return B_S = \{((\beta_{W_i}, \mu_{W_i}) | W_i \in S^o) \cup \{\beta_S\}

Function GenerateHF(B_S, W^o)
1. for i = 1 \ldots \beta_S do
2.   Choose a function h^o_i uniformly at random from $\mathcal{H}_{\bar{a}, \bar{b}, r}$
3.   return B_S = \{h_i^o | 1 \leq i \leq \beta_S\}

Function CreateHT(P, B_S, H_S)
1. for i = 1 \ldots \beta_S do
2.   Create a hash table $T_i^S$ according to $h_i^S$
3. return $T_S = \{T_i^S | 1 \leq i \leq \beta_S\}$

Function SearchHT(P, q, W_i, b; B_S, H_S; T_S)
/* $h_i^S$ denotes the level-l bucket with respect to $h_i^S$ and $q$. */
1. Obtain $\beta_{W_i}$ and $\mu_{W_i}$ from B_S;
2. R = \{W_i, C = \emptyset, i = 1;
3. while TRUE do
4.   for i = 1 \ldots \beta_{W_i} do
5.     Count collision for points in bucket $h_i^S(q)$ of hash table $T_i^S$ and add frequent points to $C$
6.     if |\{o \in C \land B_W(q) \leq c \times R\}| \geq k then
7.       return exact k-nearest neighbors of q in C under $W_i$
8.     if |C| \geq k + \gamma n then
9.       return exact k-nearest neighbors of q in C under $W_i$
10. R = c \times R, i = i + 1;
```
The storage costs for WNN search under the algorithm determines deduplicating in multiple weighted sets of been proposed to return non-optimal weighted set covers is NP-complete [36]. Thus, a number of algorithms have been designed to improve the query efficiency, query accuracy and space consumption of WLSH. Usually, it is expensive to provide a worst-case guarantee on the query accuracy. Thus, we can optionally improve the query efficiency and reduce the space consumption at the sacrifice of the query accuracy guarantee.

Following the idea, we introduce two approaches for optimizing WLSH: bound relaxation and collision threshold reduction. The goal of the bound relaxation is to reduce the space consumption, while the goal of the collision threshold reduction is to improve the query efficiency. The two approaches can be applied together.

Bound Relaxation. WLSH is based on Theorem 1. As we can see, the bounds \( R^\dagger \) and \( (cR)^\dagger \) given by Theorem 1 are independent of \( P \) and \( q \). This implies that they are probably too strict in practice. We can relax them as follows:

\[
R^\dagger = R^\dagger(v) \quad \text{(14)}
\]
\[
(cR)^\dagger = cR^\dagger(d+1-v') \quad \text{(15)}
\]

where \( T^\dagger(v) \) is the \( v \)th largest number in \( T \), \( T^{d+1-v'} \) is the \( v' \)th smallest number in \( T \), and \( T = \{ w_i / w_i' | 1 \leq i \leq d \} \), and \( 1 \leq v \leq d + 1 - v' \leq d \). Obviously, \( R^\dagger \) and \( (cR)^\dagger \) are the same as in Theorem 1 when \( v = v' = 1 \). If \( v \) and \( v' \) are set to be greater than 1, then the required number of hash tables for each weight vector tends to be reduced accordingly to Equation 14. As a result, the required total number of hash tables is likely to be reduced.

Collision Threshold Reduction. We can improve the query efficiency of WLSH by using a smaller collision threshold. The intuition is that, the data points that are close to \( q \) can be identified as frequent points earlier with a smaller collision threshold. Actually, C2LSH has already adopted the collision threshold reduction approach to improve its query efficiency [15]. In the experiments in Section 5, we will use the collision threshold \( \mu_{W_i} = X \mu_{W_i} \) for the (c,k)-WNN search under \( W_i \in S \) when collision threshold reduction is applied, where \( X = \)
5 Experiments

The experiments are targeted to answer four questions:

1. How does the space consumption of WLSH vary with various parameters? (Section 5.2.1)
2. How does the space consumption of WLSH compare with those of SL-ALSH and S2-ALSH? (Section 5.2.2)
3. How do the query efficiency and query accuracy of WLSH vary with various parameters? (Section 5.3.1)
4. How do the query efficiency and query accuracy of WLSH compare with those of SL-ALSH and S2-ALSH? (Section 5.3.2)

We focus on the $l_1$ distance and $l_2$ distance while evaluating the performance of WLSH due to space limitations. Moreover, we compare WLSH with SL-ALSH and S2-ALSH only for the $l_2$ distance since SL-ALSH and S2-ALSH are both designed for the $l_2$ distance.

5.1 Experimental Setup

5.1.1 Data Sets, Weight Vector Sets and Query Sets

The data sets, weight vector sets and query sets used in the experiments are introduced as follows.

Data Sets. We use both synthetic and real data sets. The parameter settings for the synthetic data sets are summarized in Table 3 where the default dimensionality and cardinality are underlined. For each synthetic data set, the values in each dimension are integers uniformly distributed in the range $[0, 10,000]$. The real data sets used are Sift, Ukbench, Notre and Sun. The statistics of them are summarized in Table 4. For each real data set, the values in each dimension are normalized to be integers in a given range shown in Table 4. More details about them are available in [40].

Weight Vector Sets. The data sets with the same dimensionality share the same collection of weight vector sets. For $d$-dimensional data sets, the weight vector sets used are identified by weight vector set cardinality $|S|$ and the other two parameters #Subset and #Subrange. Given $|S|$, #Subset and #Subrange, the corresponding weight vector set is the union of #Subset equal-size subsets, each of which is generated as follows. First, the range $[1, 10]$ is divided into $|S|$ equal-width subranges, denoted by $X$. Then, a set of $d$ subranges, denoted by $Y = \{Y_i | 1 \leq i \leq d \}$, are chosen uniformly at random from $X$. Finally, the values in the $i$th dimension are chosen uniformly at random from the subrange $Y_i$ for $1 \leq i \leq d$. Obviously, a number of different weight vector sets can be obtained by varying $|S|$, #Subset and #Subrange. For example, a set of uniformly random weight vectors from $[1, 10]^d$ can be obtained by setting #Subset = $|S|$ and #Subrange = 1.

Besides the uniformly random weight vector sets, we also use some other weight vector sets whose parameter settings are shown in Table 5 where the default values of $|S|$, #Subset and #Subrange are underlined.

Query Sets. Given a data set and a weight vector set, a query set is obtained in the following way. First, 50 points are chosen uniformly at random from the data set as a query point set (these points are removed from the data set at the same time). Then, 10 weight vectors are chosen uniformly at random from the weight vector set as a query weight vector set. Finally, the Cartesian product of the query point set and the query weight vector set is taken as a query set.

5.1.2 Evaluation Metrics

Following previous work [15], [41], three metrics are adopted in our evaluations.

Space Consumption. The space consumption is measured by the required total number of hash tables.

Query Efficiency. Since the $(c, k)$-WNN search is I/O intensive, the query efficiency is measured by the I/O cost averaged over a query set. Specifically, the I/O cost consists of the cost for identifying candidates and the cost for checking candidates.

Query Accuracy. As in [14], [15], [41], the query accuracy is measured by the average overall ratio, which is the mean of the overall ratios over a query set. For a query whose query point and query weight vector are $q$ and $\tilde{W}$ respectively, the overall ratio is computed by:

$$R(q, \tilde{W}) = \frac{1}{k} \sum_{i=1}^{k} \frac{D_{\tilde{W}}(o_i, q)}{D_{\tilde{W}}(o_i^*, q)}$$

where $o_1, o_2, \ldots, o_k$ are the reported approximate $k$-nearest neighbors sorted in non-decreasing order of their distances to $q$, and $o_1^*, o_2^*, \ldots, o_k^*$ are the exact $k$-nearest neighbors sorted in the same way.

5.1.3 Settings of Other Parameters

The block size is set to 4KB. The result cardinality $k$ is in the range $[10, 100]$. We set the threshold $\tau$ to 1,000 for the $l_1$ distance and 500 for the $l_2$ distance. Unless otherwise specified, the approximation ratio $c$ is in the range $\{2, 3, 4, 5, 6\}$, and its default value is 3. For a $d$-dimensional weight vector set, we set the parameters $v = v' = d/4$ when bound relaxation is applied.

2. https://github.com/DBWangGroupUNSW/nns_benchmark
5.2 Space Consumption of WLSH

5.2.1 Impacts of Various Parameters

As the space consumption of WLSH is independent of data distribution, we use only synthetic data sets to study the impacts of various parameters on it. The parameters involved are $d$, $n$, $c$, $\#\text{Subrange}$, $\#\text{Subset}$ and $|S|$. We evaluate the impact of each parameter by varying one parameter and setting the other five parameters to the default values. The experimental results are presented in Table 6 and the followings are brief explanations of them.

Impacts of $d$, $n$ and $c$. The required number of hash tables for each weight vector is directly affected by $x^\dagger$ and $y^\dagger$ rather than $d$ from Equation (11). However, we observe that, due to our weight vector set generation method, $x^\dagger$ tends to increase with $d$ and $y^\dagger$ tends to decrease with $d$ when bound relaxation is not used, while $x^\dagger$ tends to decrease with $d$ and $y^\dagger$ tends to increase with $d$ when bound relaxation is used. Therefore, according to Equation (11) the required number of hash tables for each weight vector tends to increase with $d$ when bound relaxation is not used and decrease with $d$ when bound relaxation is used. The required total number of hash tables has the same trend as the required number of hash tables for each weight vector.

The required number of hash tables for each weight vector tends to increase with $n$ and decrease with $c$ according to Equation (11). Thus, the required total number of hash tables tends to increase with $n$ and decrease with $c$.

Impacts of $\#\text{Subrange}$, $\#\text{Subset}$ and $|S|$. According to our weight vector set generation method, a weight vector set is the union of $\#\text{Subset}$ equal-size subsets and each of the subsets contains similar weight vectors. As $\#\text{Subrange}$ increases, the weight vectors in the same subset become more similar to each other and tend to share more hash tables so as to minimize the required total number of hash tables. Thus, the required total number of hash tables tends to decrease with $\#\text{Subrange}$. Since the weight vectors in different subsets are quite dissimilar, they can hardly share hash tables. Consequently, the required total number of hash tables tends to increase with $\#\text{Subset}$.

The impact of $|S|$ depends on $\#\text{Subrange}$. When $\#\text{Subrange}$ is sufficiently large, the weight vectors in the same subset can share the same group of hash tables, no matter how big $|S|$ is. In this case, the required total number of hash tables tends to remain relatively stable with the increase of $|S|$. However, when $\#\text{Subrange}$ is not so large (e.g., $\#\text{Subrange}$ is set to the default value), the weight vectors in the same subset can not share the same group of hash tables. In this case, the required total number of hash tables tends to increase with $|S|$.

Additionally, we can draw the following conclusions from Table 6. First, the space consumption of WLSH can be significantly reduced by using bound relaxation. Actually, without using bound relaxation, WLSH can even degenerate into the naive method mentioned in Section 2.4 in a small fraction of cases (marked in bold in Table 6). Second, the space consumption of WLSH for the $l_1$ distance is higher than that for the $l_2$ distance in almost all cases. The fundamental reason is that, the required number of hash tables for each weight vector tends to be larger for the $l_1$ distance than for the $l_2$ distance according to Equation (11).

Though the space consumption of WLSH can also be reduced by increasing $c$ from Table 6 we experimentally demonstrate the necessity of introducing bound relaxation to WLSH in Appendix F.1.

5.2.2 Comparison with SL-ALSH and S2-ALSH

We evaluate the space consumptions of SL-ALSH and S2-ALSH first. The space complexities of SL-ALSH and S2-ALSH are discussed in Appendix A. It is not explicitly stated in [25] how to create hash tables at different radii for SL-ALSH and S2-ALSH. That seems not an easy task. Here we simply assume that hash tables are created at a single radius $R = 1000$ for both SL-ALSH and S2-ALSH (as a consequence, the space consumptions of SL-ALSH and S2-ALSH are underestimated). Then the required total numbers of hash tables for SL-ALSH and S2-ALSH are computed by $L_{SL} = n^{p_{SL}}$ and $L_{S2} = n^{p_{S2}}$ respectively, where $p_{SL}$ and $p_{S2}$ are shown in Appendix A. The settings of $d$, $n$, $c$, $\#\text{Subrange}$, $\#\text{Subset}$ and $|S|$ are the same as in Section 5.2.1. The results are presented in Table 7.

It can be seen from Table 7 that the required total numbers of hash tables for SL-ALSH and S2-ALSH increase with $n$ and decrease with $c$, and they are not significantly affected by the other four parameters. We can calculate that $p_{SL}$ and $p_{S2}$ are both greater than 0.98 (very close to 1) when $c \leq 3$ and greater than 0.75 when $c \leq 6$. This indicates that the required total numbers of hash tables for SL-ALSH and S2-ALSH can increase very fast with $n$. In contrast, the required total number of hash tables for WLSH grows logarithmically with $n$. Combining Section 5.2.1 and Table 7 we conclude that the space consumptions of SL-ALSH and S2-ALSH are more sensitive to the data set, while the space consumption of WLSH is more sensitive to the weight vector set.

5.3 Query Efficiency and Query Accuracy of WLSH

Since bound relaxation is effective in reducing the space consumption of WLSH, we apply it to WLSH in this section.

5.3.1 Impacts of Various Parameters

Both synthetic and real data sets are used to study the impacts of various parameters on the query efficiency and query accuracy of WLSH. The evaluation on synthetic data sets involves the parameters $d$, $n$, $c$, $\#\text{Subrange}$, $\#\text{Subset}$ and $|S|$, and the evaluation on real data sets involves the parameters $c$, $\#\text{Subrange}$, $\#\text{Subset}$ and $|S|$. As in Section 5.2.1, the impact of each parameter is evaluated by varying one parameter and setting the other parameters to the default values.

The results on synthetic data sets for WLSH using collision threshold reduction are shown in Fig. 1 and the results on real data sets for WLSH using collision threshold reduction are shown in Figs. 5-7 in Appendix F.2. Additionally, Figs. 23-27 in Appendix F.2 show the results on synthetic data sets for WLSH without using collision threshold reduction. It can be seen from Fig. 1 and Figs. 23-27 in Appendix F.2 that by adopting collision threshold reduction, the query efficiency of WLSH can be improved significantly while the query accuracy of WLSH is still high. Thus, we apply collision threshold reduction to WLSH in the subsequent
TABLE 6
Space Consumption of WLSH

| $l_1$ distance | $l_2$ distance |
|----------------|----------------|
| $d$ | 100 | 200 | 400 | 800 | 1.6k |
| $\beta_g$ | 453,613 | 1,224,539 | 2,097,571 | 2,160,000 | 2,160,000 |
| $\beta_{g'}$ | 106,960 | 102,799 | 98,310 | 94,243 | 94,243 |
| $n$ | 1,641,456 | 1,643,438 | 2,097,571 | 2,226,029 | 2,239,799 |
| $\beta_{g''}$ | 89,314 | 93,846 | 98,310 | 102,866 | 106,546 |
| $c$ | 2 | 3 | 4 | 5 | 6 |
| $\beta_g$ | 4,150,000 | 2,097,571 | 2,251,398 | 158,190 | 128,785 |
| $\beta_{g'}$ | 197,636 | 98,310 | 72,993 | 62,101 | 56,789 |

#Subrange
5 | 10 | 20 | 50 | 100 |
$\beta_g$ | 2,160,000 | 2,160,000 | 2,160,000 | 2,160,000 | 2,160,000 |
$\beta_{g'}$ | 136,804 | 108,279 | 89,310 | 88,273 | 87,174 |
$\beta_{g''}$ | 31,728 | 52,781 | 98,310 | 231,124 | 462,857 |
#Subset
5 | 10 | 20 | 50 | 100 |
$\beta_g$ | 1,846,151 | 2,036,839 | 2,097,571 | 2,146,358 | 2,156,237 |
$\beta_{g'}$ | 31,728 | 52,781 | 98,310 | 231,124 | 462,857 |
$\beta_{g''}$ | 1,430,804 | 1,266,204 | 2,097,571 | 2,240,547 | 3,414,455 |

Besides, the reason why the I/O cost of WLSH decreases with $c$ and the average overall ratio of WLSH increases with $c$ is that, as $c$ increases, the collision threshold gets smaller such that the I/O cost for collision counting tends to decrease and the distances from a query point to the identified frequent points tend to increase.

From Fig. 1 and Figs. 47 in Appendix F2, we can also see that the I/O cost of WLSH slightly increases with $k$. The reason is that more frequent points need to be identified by doing collision counting as $k$ increases. In addition, the I/O cost of WLSH for the $l_1$ distance is higher than that for the $l_2$ distance in most cases. The reason is that the collision threshold tends to be larger for the $l_1$ distance than for the $l_2$ distance such that the I/O cost for collision counting tends to be higher for the $l_1$ distance than for the $l_2$ distance.

5.3.2 Comparison with SL-ALSH and S2-ALSH

TABLE 7
Space Consumptions of SL-ALSH and S2-ALSH ($R = 1000$)

| $l_2$ distance | $l_2$ distance |
|----------------|----------------|
| $d$ | 100 | 200 | 400 | 800 | 1.6k |
| $\beta_g$ | 336,626 | 325,715 | 322,001 | 320,141 | 317,348 |
| $\beta_{g'}$ | 581,038 | 325,715 | 319,349 | 317,624 | 316,363 |
| $n$ | 100k | 200k | 400k | 800k | 1.6m |
| $\beta_g$ | 82,405 | 162,884 | 322,001 | 636,300 | 1,258,699 |
| $\beta_{g'}$ | 81,793 | 161,618 | 319,349 | 631,015 | 1,246,850 |
| $c$ | 2 | 3 | 4 | 5 | 6 |
| $\beta_g$ | 387,551 | 322,001 | 192,819 | 76,763 | 20,945 |
| $\beta_{g'}$ | 387,551 | 319,349 | 189,406 | 71,911 | 18,093 |
| $\beta_{g''}$ | 322,571 | 321,165 | 322,001 | 321,098 | 322,221 |
| $\beta_{g'}$ | 319,832 | 318,586 | 319,349 | 318,396 | 324,301 |
| $\beta_{g''}$ | 324,596 | 322,799 | 322,001 | 323,635 | 323,740 |
| $\beta_{g'}$ | 321,727 | 320,004 | 319,349 | 320,775 | 320,902 |
| $\beta_{g''}$ | 322,793 | 322,508 | 322,001 | 321,816 | 323,251 |
| $\beta_{g'}$ | 320,117 | 319,766 | 319,349 | 319,174 | 320,545 |

$^*$ $\beta_g$ is the required total number of hash tables for SL-ALSH and S2-ALSH.
Two groups of experiments are carried out on real data sets. In the first group of experiments, we vary the parameters c, #Subrange, #Subset and |S| in the same way as in Section 5.3.1. Due to space limitations, the results are shown in Figs. 8, Appendix 2. In summary, the average overall ratio of WLSH is smaller than those of SL-ALSH and S2-ALSH in 120/160 and 115/160 of cases respectively. In the second group of experiments, we use uniformly random weight vector sets and set c to 8 so that the space consumptions of WLSH, SL-ALSH and S2-ALSH are similar and moderate. The experimental results are presented in Table 8. In the table, the average overall ratio of WLSH is smaller than those of SL-ALSH and S2-ALSH in 5/8 and 5/8 of cases respectively. As a whole, WLSH performs better than SL-ALSH and S2-ALSH in the two groups of experiments. This implies that the asymmetric LSH families for SL-ALSH and S2-ALSH are less effective than the weighted LSH families and derived weighted LSH families for WLSH. Actually, SL-ALSH and S2-ALSH are not so appropriate as WLSH for solving our problem since they create hash tables regardless of the weight vector set due to their asymmetric LSH families.

6 RELATED WORK

6.1 Locality-Sensitive Hashing for Approximate Nearest Neighbor Search

Nearest neighbor search arises in a wide variety of applications, such as product recommendation [22], image retrieval [42] and clustering [43]. Locality-Sensitive Hashing (LSH) is a popular approach for approximate nearest neighbor search in high-dimensional spaces [10, 11]. Several important LSH schemes are briefly reviewed as follows.

The first LSH scheme is E2LSH proposed by Indyk et al. [12]. It can answer nearest neighbor queries in sublinear time. However, it needs to create hash tables at a series of radii, which is space-consuming. The multi-probe LSH scheme proposed by Lh et al. reduces the required number of hash tables by checking not only the data points that fall in the same bucket as the query point but also the data points that fall in the nearby buckets [13]. LSB-tree/LSB-forest proposed by Tao et al. is the first LSH scheme that avoids creating hash tables at different radii by exploiting virtual rehashing [14]. Since LSB-tree/LSB-forest uses compound hash functions to create hash tables, the number of hash buckets to read increases exponentially with the search radius, which is undesirable. To overcome the drawback, Gan et al. propose C2LSH where each hash table is created by using a single LSH function [15]. Combining the
virtual rehashing and collision counting techniques, C2LSH is proven to be better than LSB-tree/LSB-forest. Recently, QALSH/QALSH proposed by Huang et al. achieves better performance than C2LSH by introducing query-aware LSH families and a query-aware bucket partition strategy [16].

6.2 SL-ALSH, S2-ALSH and LazyLSH

In the following, we cover the prior methods related to our work. They all take the LSH approach.

**SL-ALSH and S2-ALSH.** The problem studied in [25] is most similar to ours. It is to support efficient approximate nearest neighbor search for all possible weighted distance functions with respect to the $l_2$ distance. SL-ALSH and S2-ALSH [25] are proposed to address the problem. The novelty of SL-ALSH and S2-ALSH is that they each introduce an asymmetric LSH family to create hash tables and process queries. Each of the asymmetric LSH families involves two mappings which can map data points and query points to different hyperspheres respectively. The mapping for data points is independent of any weighted distance function, while the mapping for query points is sensitive to the associated weighted distance functions. Since the two asymmetric LSH families target the $l_2$ distance, they are not available for the $l_p$ distance for $p \in (0, 2)$.

**LazyLSH.** The work of [41] studies how to support efficient approximate nearest neighbor search under the distance function with respect to the $l_p$ distance, where $p$ is taken as a variable. The method proposed for solving this problem is called LazyLSH [41]. Briefly, LazyLSH first creates a group of hash tables using an LSH family originally proposed for the $l_1$ distance, and then processes all incoming queries with various values of $p$ over the hash tables. The motivation behind LazyLSH is that if two points are close under the $l_p$ distance function, then they are likely to be close under the $l_2$ distance function, where $p_1 \neq p_2$. The idea of LazyLSH is not available for solving our problem since two close points under a weighted distance function can be far away under another weighted distance function.

7 Conclusion

In this paper, we consider the important problem of supporting efficient approximate nearest neighbor search for multiple weighted distance functions in high-dimensional spaces. The WLSH method is proposed to solve the problem for the $l_p$ distance with $p \in (0, 2)$. Remarkably, it is the first method that can solve the problem for the $l_p$ distance for $p \in (0, 2)$. Theoretically, WLSH can guarantee the efficiency of processing queries and the accuracy of query results while minimizing the space consumption. To achieve better practical performance, WLSH can also make reasonable trade-offs among the query efficiency, query accuracy and space consumption. Extensive experiments are carried out on both synthetic and real data sets, and the results verify the effectiveness and efficiency of WLSH. However, we observe that the performance of WLSH degrades as $p$ decreases. A promising direction for future work is to base WLSH on QALSH/QALSH [16] instead of on C2LSH so that the performance of WLSH could be further improved. We believe that this can be done without the need of making many modifications to our main techniques in this paper.

**APPENDIX A**

**TIME AND SPACE COMPLEXITIES OF SL-ALSH AND S2-ALSH**

Both the SL-ALSH and S2-ALSH methods [25] can be used to address the problem studied in the paper. Here we give the time complexities of SL-ALSH and S2-ALSH for processing queries and the space complexities of SL-ALSH and S2-ALSH for storing hash tables and original data.

For SL-ALSH and S2-ALSH, suppose $o, q \in [0, V]^d \subset \mathbb{R}^d$ for all $o \in P$ and $q$ for $0 < V \leq \pi$. Otherwise, we can shift and/or rescale $o \in P$ and $q$ without changing the query results. Further, suppose $||\tilde{W}_i||_1 = 1$ for all $\tilde{W}_i \in S$. Otherwise, we can also rescale $\tilde{W}_i \in S$ without changing the query results. It is not explicitly stated in [25] how SL-ALSH and S2-ALSH create hash tables at different radii.

Thus, we simply assume that they only create hash tables at a radius $R$ for all $\tilde{W}_i \in S$. Let $\eta_{V_i} = \sqrt{d}||\tilde{W}_i||_2$ for $\tilde{W}_i \in S$. According to Theorem 3 in [25] and Definition 4, the time and space complexities of SL-ALSH are $O(n^p_{SL} \cdot d \log n)$ and $O(nd + n^{1+p_{SL}})$ respectively, where

$$\rho_{SL} = \min_{w, V} \left( \max_{\tilde{W}_i \in S} \left( \ln \left( \frac{P_2 \left( \frac{2\eta_{V_i}}{\eta_{V_i}} - 2 + R \right)}{P_2 \left( \frac{2\eta_{V_i}}{\eta_{V_i}} - 2 + cR - \frac{1}{12} V^4 \right)} \right) \right) \right)$$

According to Theorem 4 in [25] and Definition 4, the time and space complexities of S2-ALSH are $O(n^p_{2SL} \cdot d \log n)$ and $O(nd + n^{1+p_{2SL}})$ respectively, where

$$\rho_{S2} = \min_{V} \left( \max_{\tilde{W}_i \in S} \left( \ln \left( \frac{1 - \frac{1}{\pi} \arccos \left( \frac{1 - \frac{1}{\eta_{V_i}} R}{\eta_{V_i}} \right) }{1 - \frac{1}{\pi} \arccos \left( \frac{1 - \frac{1}{\eta_{V_i}} R + \frac{1}{12} V^4}{\eta_{V_i}} \right) } \right) \right) \right)$$

Please note that in Equation (17) $P_2(r) = \int_0^r \frac{1}{\pi} F_2 \left( \frac{1}{\pi} \right) \left( 1 - \frac{r}{\pi} dt \right)$, where $F_2(\cdot)$ is the PDF of the absolute value of the standard normal distribution. It can be verified that $P_2(r)$ is smaller than 1 and decreases with $r$. Both SL-ALSH and S2-ALSH require that $cR - \frac{1}{12} V^4 > R$. Thus, $0 < \rho_{SL}, \rho_{S2} < 1$. The required total number of hash tables for SL-ALSH is $n^p_{SL}$, and the required total number of hash tables for S2-ALSH is $n^p_{S2}$.

**APPENDIX B**

**TRADITIONAL LSH FAMILIES AND WEIGHTED LSH FAMILIES FOR HAMMING DISTANCE AND ANGULAR DISTANCE**

In the following, we give the distance functions with respect to the Hamming distance and angular distance.

**Hamming Distance:** Let $\vec{x} = (x_1, x_2, \ldots, x_d) \in \{0, 1\}^d$ and $\vec{y} = (y_1, y_2, \ldots, y_d) \in \{0, 1\}^d$. The distance function with respect to the Hamming distance is

$$D(x, y) = \sum_{i=1}^{d} |x_i - y_i|$$

(19)
Angular Distance: Let \( \vec{x} = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d \) and \( \vec{y} = (y_1, y_2, \ldots, y_d) \in \mathbb{R}^d \). The distance function with respect to the angular distance is
\[
D(x, y) = \arccos \left( \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\|_2 \times \|\vec{y}\|_2} \right) \tag{20}
\]

In Table 3 we present a classical traditional LSH family for the Hamming distance and a classical traditional LSH family for the angular distance. The corresponding weighted LSH families are shown in Table 10.

APPENDIX C

PROOF OF THEOREM 1

Proof

(1) Let \( \vec{x} = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d \), \( \vec{y} = (y_1, y_2, \ldots, y_d) \in \mathbb{R}^d \), \( \vec{w} \), and \( \vec{w}_\gamma \) are max1≤i≤d (\( w_i \|\vec{w}\|_2 \)) and \( \vec{w}_\gamma = \min1≤i≤d (\|\vec{w}_i\|_2) \). For the \( \ell_\gamma \) distance, we have
\[
D_{\vec{w}}(x, y) = \sqrt{\sum_{i=1}^{d} w_i^2 (x_i - y_i)^2} \quad \text{and} \quad D_{\vec{w},\gamma}(x, y) = \sqrt{\sum_{i=1}^{d} \gamma_i^2 (x_i - y_i)^2}.
\]

First, we need to prove \( R^1 = R_{\vec{w}}^\gamma \). Under the constraint of \( D_{\vec{w}}(x, y) \), it is equivalent to prove \( R_{\vec{w}}^\gamma \) is the maximum \( \sum_{i=1}^{d} w_i^2 |x_i - y_i|^2 \leq \sum_{i=1}^{d} \gamma_i^2 |x_i - y_i|^2 \) under the constraint of \( R_{\vec{w}}^\gamma \) is the maximum \( \sum_{i=1}^{d} \gamma_i^2 |x_i - y_i|^2 \). From (20), we have that \( R_{\vec{w}}^\gamma \) is the maximum \( \sum_{i=1}^{d} \gamma_i^2 |x_i - y_i|^2 \) under the constraint of \( R_{\vec{w}}^\gamma \) is the maximum \( \sum_{i=1}^{d} \gamma_i^2 |x_i - y_i|^2 \).

Next, we need to prove \( cR^1 = cR_{\vec{w}}^\gamma \). Under the constraint of \( D_{\vec{w}}(x, y) \), it is equivalent to prove \( cR_{\vec{w}}^\gamma \) is the maximum \( \frac{\sum_{i=1}^{d} w_i^2 |x_i - y_i|^2}{\sum_{i=1}^{d} \gamma_i^2 |x_i - y_i|^2} \) under the constraint of \( cR_{\vec{w}}^\gamma \) is the maximum \( \frac{\sum_{i=1}^{d} \gamma_i^2 |x_i - y_i|^2}{\sum_{i=1}^{d} \gamma_i^2 |x_i - y_i|^2} \).

APPENDIX D

PROOF OF LEMMA 1

Please note that \( X = \mathbb{R}^d \) for the \( \ell_\gamma \) distance in the paper. Given any integer \( i \geq 1 \), define a traditional LSH family \( \mathcal{H}_{\vec{a},\vec{b}} = \{ h_{\vec{a},\vec{b}} : X \to U \} \), where
\[
h_{\vec{a},\vec{b}}(x) = \left\lfloor \frac{\vec{a} \cdot \vec{x} + bi}{w} \right\rfloor,
\]
and \( \vec{a}, \vec{b} \) and \( w \) are the same as in Equation 2. The proof of Lemma 1 is based on the proof of Observation 1 in [14].

In brief, the proof of Observation 1 in [14] shows that the probability of any two points in \( X \) colliding under an LSH function from \( \mathcal{H}_{\vec{a},\vec{b}} \) is not affected by \( i \).

Proof

(1) The weighted LSH family obtained from \( \mathcal{H}_{\vec{a},\vec{b}} \) is \( \mathcal{H}_{\vec{a},\vec{b},\vec{w}} = \{ h'_{\vec{a},\vec{b},\vec{w}} : X \to U \} \), where
\[
h'_{\vec{a},\vec{b},\vec{w}}(x) = \left\lfloor \frac{\vec{a} \cdot (\vec{w} \cdot \vec{x}) + bi}{w} \right\rfloor,
\]
and \( \vec{a}, \vec{b} \) and \( w \) are the same as in Equation 2. According to the proof of Observation 1 in [14], the probability of any two points in \( \vec{W} \circ \vec{x} \) colliding under an LSH function from \( \mathcal{H}_{\vec{a},\vec{b},\vec{w}} \) is not affected by \( i \). Based on this, we can have that \( \mathcal{H}'_{\vec{a},\vec{b},\vec{w}}(x, y) = P_{\vec{a},\vec{b},\vec{w}}(x, y) \)-sensitive for any \( 0 < x < y \), given the facts that \( \mathcal{H}_{\vec{a},\vec{b},\vec{w}} \) introduced in Section 5.1 is \( \{ x, y, P_{\vec{a},\vec{b},\vec{w}}(x), P_{\vec{a},\vec{b},\vec{w}}(y) \} \)-sensitive for any
TABLE 9
Traditional LSH Families for Hamming Distance and Angular Distance

| Distance Measure | \( h(x) \) | PMF (or PDF) | \( P(\cdot) \) |
|------------------|---------|-------------|---------------|
| Hamming distance | \( h_x(k) = x_k, k \in \{1, 2, \ldots, d\} \) | \( p(k) = \frac{1}{2} \) for \( k \in \{1, 2, \ldots, d\} \). | \( P_f(r) = 1 - \frac{r}{2} \) |
| Angular distance | \( h_{\sigma}(x) = \text{sign}(\sigma \cdot x), \sigma = (u_1, u_2, \ldots, u_d) \in \mathbb{R}^d \) | \( p(\sigma) = \prod_{i=1}^d f_2(u_i) \), where \( f_2 \) is the PDF of the 2-stable distribution (i.e., the standard normal distribution). | \( P_{\theta}(r) = 1 - \frac{r}{\pi} \) |

\( \sigma = (x_1, x_2, \ldots, x_d) \)

\( \sigma = (x_1, x_2, \ldots, x_d) \) and \( \mathbb{W} = (w_1, w_2, \ldots, w_d) \)

\( 0 < x < y \) and \( H'_{\tilde{a}, b, \mathbb{W}} \) is the same as \( H_{\tilde{a}, b, \mathbb{W}} \) when \( i = 1 \). By letting \( i = c^{[\log r_{\max/min}]_{x > y}} \) and \( \mathbb{W} = \mathbb{W}_o \) for \( H'_{\tilde{a}, b, \mathbb{W}} \), we obtain that \( H_{\tilde{a}, b, \mathbb{W}} \) is \( (x, y, P_{\tilde{a}, b, \mathbb{W}}(x), P_{\tilde{a}, b, \mathbb{W}}(y)) \)-sensitive for any \( 0 < x < y \).

(2). Let \( l \in \{c, c^2, \ldots, c^{[\log r_{\max/min}]_{x > y}}\} \). From Equation \( H^{(l)}_{\tilde{a}, b, \mathbb{W}}(\mathbb{X}) \rightarrow \mathbb{U} \) and \( H^{(l)}_{\tilde{a}, b, \mathbb{W}}(\mathbb{X}) \rightarrow \mathbb{U} \),

\[
H^{(l)}_{\tilde{a}, b, \mathbb{W}}(\mathbb{X}) = \left\{ \begin{array}{ll}
\bar{a} \cdot (\mathbb{W}_o \cdot \mathbb{X}) + \left( \frac{w}{l} \right) & \text{if } \mathbb{X} \notin \mathbb{W}_o \\
\bar{a} \cdot (\mathbb{W}_o \cdot \mathbb{X}) + \left( \frac{w}{l} \right) & \text{if } \mathbb{X} \in \mathbb{W}_o
\end{array} \right.
\]

\( f = c^{[\log r_{\max/min}]_{x > y}} \) and \( \bar{a}, b \) and \( w \) are the same as in Equation \( H^{(l)}_{\tilde{a}, b, \mathbb{W}} \).

\( f = c^{[\log r_{\max/min}]_{x > y}} \) and \( \bar{a}, b \) and \( w \) are the same as in Equation \( H^{(l)}_{\tilde{a}, b, \mathbb{W}} \).

APPENDIX F
MORE EXPERIMENTAL RESULTS

F.1 Necessity of Bound Relaxation

The experimental results in Table 6 indicate that the space consumption of WLSH can be reduced by setting the approximation ratio \( c \) to a larger value. One may doubt the necessity of introducing bound relaxation to WLSH. Here we demonstrate it by using uniformly random weight vector sets and varying \( c \) in the range \( \{5, 7, 9, 11, 13\} \). The experiments are conducted on real data sets, and the results are shown in Table 11. As it can be seen from the table, when bound relaxation is not used, the required total number of hash tables for WLSH decreases slower with the increase of \( c \). Worse still, it is still very large when \( c = 13 \). This implies that it is limited to reduce the space consumption of WLSH only by increasing \( c \). In contrast, when bound relaxation is used, the required total number of hash tables for WLSH is acceptable if \( c \geq 7 \). Thus, it can be concluded that the approach of bound relaxation is necessary for reducing the space consumption of WLSH.

F.2 Query Efficiency and Query Accuracy of WLSH

Figs. 23 show the query efficiency and query accuracy of WLSH on synthetic data sets when not using collision

APPENDIX E
PROOF OF THEOREM 2

Proof:

(1). Theorem 2(1) can be directly obtained from Section 3.2 and Lemma 1(1) (it is easy to know that \( 0 < x^i < y^i \) implies \( 0 < x < y \) such that Lemma 1(1) is available).

(2). Define \( P_{l, \tilde{a}, \mathbb{W}_o}(r) = P_{l, \tilde{a}, \mathbb{W}_o}(r) \) for \( r > 0 \). According to the proof of Lemma 1(2), \( H^{(l)}_{\tilde{a}, b, \mathbb{W}_o} \) is \( (x, y, P_{\tilde{a}, b, \mathbb{W}_o}(x), P_{\tilde{a}, b, \mathbb{W}_o}(y)) \)-sensitive for any \( 0 < x < y \).

As a result, \( H^{(l)}_{\tilde{a}, b, \mathbb{W}_o}(x) \rightarrow \mathbb{U} \) and \( (y) \rightarrow y \) from Theorem 1(1). Since \( r_{\max/min} \geq r_{\max/min} \), Theorem 2(2) holds.
threshold reduction. Figs. 17 show the query efficiency and query accuracy of WLSH on real data sets when using collision threshold reduction. Figs. 18 compare the average overall ratios of WLSH, SL-ALSH and S2-ALSH on real data.

The parameter settings for SL-ALSH and S2-ALSH in Section 5.3.2 are shown in Table 12, where \(w\) and \(V\) are the same as in Equations 17 and 18 and \(m\) is the number of ALSH functions to form a compound hash function for creating a hash table. We vary \(m\) to obtain multiple average overall ratios and take the smallest one as the final result. Please note that it is almost impossible to find the optimal parameter settings for SL-ALSH and S2-ALSH in terms of query efficiency and query accuracy because they vary with the data set and weight vector set.

### TABLE 11

| \(S_b\) | \(c\) | 5 | 7 | 9 | 11 | 13 |
|---|---|---|---|---|---|---|
| \(\beta_S\) | 1,500,000 | 1,280,000 | 1,180,000 | 1,120,000 | 1,080,000 |
| \(\beta_S^*\) | 451,165 | 15,651 | 4,965 | 2,562 | 2,254 |
| \(\beta_S\) | 1,310,000 | 1,290,000 | 1,185,000 | 1,125,000 | 1,085,000 |
| \(\beta_S^*\) | 467,427 | 16,450 | 5,123 | 2,577 | 2,252 |
| \(\beta_S\) | 1,400,000 | 1,195,000 | 1,100,000 | 1,045,000 | 1,010,000 |
| \(\beta_S^*\) | 304,462 | 11,230 | 4,314 | 2,995 | 2,103 |
| \(\beta_S\) | 1,265,000 | 1,080,000 | 995,000 | 945,000 | 910,000 |
| \(\beta_S^*\) | 1,262,616 | 26,658 | 2,754 | 2,252 | 1,969 |

### TABLE 12

| \(S_b\) | \(c\) | 5 | 7 | 9 | 11 | 13 |
|---|---|---|---|---|---|---|
| \(\beta_S\) | 840,000 | 720,000 | 665,000 | 635,000 | 615,000 |
| \(\beta_S^*\) | 394,355 | 17,729 | 3,122 | 1,761 | 1,591 |
| \(\beta_S\) | 845,000 | 725,000 | 670,000 | 640,000 | 620,000 |
| \(\beta_S^*\) | 405,239 | 18,307 | 3,712 | 1,767 | 1,602 |
| \(\beta_S\) | 785,000 | 67,000 | 625,000 | 595,000 | 575,000 |
| \(\beta_S^*\) | 278,013 | 11,813 | 2,859 | 1,639 | 1,487 |

\(\beta_S\) is the required total number of hash tables when not using bound relaxation. \(\beta_S^*\) is the required total number of hash tables when using bound relaxation.

### ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China under grant NOs 61832003, U1811461, and 61732003.

### REFERENCES

[1] A. Andoni, “Nearest neighbor search: the old, the new, and the impossible,” Ph.D. dissertation, 2009.
[2] N. Bhatia and Vandana, “Survey of nearest neighbor techniques,” CoRR, 2010. [Online]. Available: http://arxiv.org/abs/1007.0085
[3] H. Samet, Foundations of multidimensional and metric data structures. Morgan Kaufmann, 2006.
[4] J. L. Bentley, "Multidimensional binary search trees used for associative searching," Commun. ACM, 1975.
[5] K. L. Cheung and A. W. Fu, “Enhanced nearest neighbour search on the r-tree,” SIGMOD Rec., vol. 27, no. 3, pp. 16–21, 1998.
[6] L. Chen, “Curse of dimensionality,” in Encyclopedia of Database Systems, 2009, pp. 545–546.
[7] M. Muja and D. G. Lowe, “Fast approximate nearest neighbors with automatic algorithm configuration,” in VISAPP, 2009.
[8] N. Ailon and A. Chazelle, “Approximate nearest neighbors and the fast johnson-lindenstrauss transform,” in Proceedings of the 38th Annual ACM Symposium on Theory of Computing, 2006, pp. 557–563.
[9] M. Muja and D. G. Lowe, “Scalable nearest neighbor algorithms for high dimensional data,” IEEE Trans. Pattern Anal. Mach. Intell., 2014.
[10] A. Andoni, P. Indyk, and I. P. Razenshteyn, “Approximate nearest neighbor search in high dimensions,” CoRR, 2018. [Online]. Available: http://arxiv.org/abs/1806.09823
[11] J. Wang, H. I. Shen, J. Song, and J. Ji, “Hashing for similarity search: A survey,” CoRR, 2014. [Online]. Available: http://arxiv.org/abs/1408.2927
[12] P. Indyk and R. Motwani, “Approximate nearest neighbors: Towards removing the curse of dimensionality,” in STOC, 1998.
[13] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, “Multi-probe LSH: efficient indexing for high-dimensional similarity search,” in VLDB, 2007.
[14] Y. Tao, K. Yi, C. Sheng, and P. Kalnis, “Efficient and accurate nearest neighbor and closest pair search in high-dimensional space,” TODS, 2010.
[15] J. Gan, J. Feng, Q. Fang, and W. Ng, “Locality-sensitive hashing scheme based on dynamic collision counting,” in SIGMOD, 2012.
[16] Q. Huang, J. Feng, Q. Fang, W. Ng, and W. Wang, “Query-aware locality-sensitive hashing scheme for lps norm,” VLDB J., 2017.
[17] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, “Locality-sensitive hashing scheme based on p-stable distributions,” in SoCG, 2004.
[18] R. C. de Amorim, “A survey on feature weighting based k-means algorithms,” J. Classification, 2016.
[19] S. Debnath, N. Ganguly, and P. Mitra, “Feature weighting in content based recommendation system using social network analysis,” in WWW, 2008.
[20] C. F. Eick, A. Rouhana, A. Bagherjeiran, and R. Vialta, “Using clustering to learn distance functions for supervised similarity assessment,” Engineering Applications of Artificial Intelligence, 2006.
[21] C. Hwang, Y. Su, and K. Tseng, “Using genetic algorithms for personalized recommendation,” in ICCCI, 2010.
[22] N. Liu, “Comparison of content-based music recommendation using different distance estimation methods,” Appl. Intell., 2013.
[23] J. J. McAuley, C. Targett, Q. Shi, and A. van den Hengel, “Image-based evaluations of feature weighting methods for a class of lazy learning algorithms,” Artif. Intell. Rev., 1997.
[24] Y. Lei, Q. Huang, M. S. Kankanhalli, and A. K. H. Tung, “Sublinear time nearest neighbor search over generalized weighted space,” in ICML, 2019.
[25] C. C. Aggarwal, V. Hinneburg, and D. A. Keim, “On the surprising behavior of distance metrics in high dimensional spaces,” in ICDT, 2001, pp. 420–434.
[26] V. Hinneburg, C. C. Aggarwal, and D. A. Keim, “What is the nearest neighbor in high dimensional spaces?” in VLDB, 2000, pp. 506–515.
[27] P. Howarth and S. M. Rüger, “Fractional distance measures for content-based image retrieval,” in ECIR, 2005, pp. 447–456.
Fig. 2. Query efficiency and query accuracy of WLISH on synthetic data when not using collision threshold reduction, $l_1$ distance.
Fig. 3. Query efficiency and query accuracy of WLSH on synthetic data when not using collision threshold reduction, $l_2$ distance
Fig. 4. Query efficiency and query accuracy of WLSH on real data when using collision threshold reduction, $\ell_1$ distance, $k = 10$

Fig. 5. Query efficiency and query accuracy of WLSH on real data when using collision threshold reduction, $\ell_1$ distance, $k = 100$
using collision threshold reduction.

**Fig. 6.** Query efficiency and query accuracy of WLSH on real data when using collision threshold reduction, $l_2$ distance, $k = 10$

**Fig. 7.** Query efficiency and query accuracy of WLSH on real data when using collision threshold reduction, $l_2$ distance, $k = 100$
Fig. 8. Comparison of average overall ratios of WLSH, SL-ALSH and S2-ALSH, $k = 10$
Fig. 9. Comparison of average overall ratios of WLSH, SL-ALSH and S2-ALSH, $k = 100$