Query-Aware Locality Sensitive Hashing for Similarity Search Problems

Kejing Lu
Graduate School of Information Science and Technology
Hokkaido University

A thesis submitted for the degree of
doctor of information engineering(PhD)

Abstract

Similarity search is an old but fundamental research topic which has various applications in database, data mining, information retrieval and machine learning. Although the goal of similarity search, that is, finding most similar points of issued queries in the dataset given a specified measure, is quite straightforward, it is very challenging to solve this problem efficiently due to the following two reasons. (1) In recent years, as the data size increases rapidly, we need to deal with up to billion-scale datasets, on which many traditional techniques lose the effectiveness. (2) Due to the phenomenon called the curse of dimensionality, the distances among data points become much closer as the dimension increases, which makes most of tree structures perform even worse than the linear scan. In order to overcome these two obstacles, many researchers have devoted to find more efficient techniques in the past two decades and proposed various algorithms.

Although these algorithms vary much, the following two performance metrics always attract particular attention in their designs: one is the accuracy, which is often indicated by the percentage of true points which are successfully found, also called, the recall rate, and the other one is the efficiency which is often indicated by the running time of searching. In order to make the accuracy more controllable, researchers have developed some techniques with theoretical guarantees to satisfy specified success probabilities. Among these techniques, Locality Sensitive Hashing (LSH) draws a particular interest due to its attractive query performance and robust probability guarantee. The basic idea of LSH is to build a group of projected vectors and select the most promising candidates only from the projection information of those vectors. Such an idea becomes the starting point of the research work introduced in this paper.

Since the solutions of similarity search problems vary highly over different spaces and different measures, in this paper, we particularly focus on the most important two situations: the Euclidean space equipped with ℓ_2 metric and the inner product space. In practice, many applications are intimately related to either of these two situations. Accordingly, this thesis is divided into two major parts.

In the first part, we focus on the ℓ_2 metric and aim to solve approximate nearest neighbor search problems. For this purpose, we propose two disk-based LSH variants called VHP and R2LSH, which are highly inspired by the concept of query-aware search window. By exploiting more accurate projection information of the generated one-dimensional projected vectors, these two methods build more effective query-centric search regions in the projected spaces. Specifically, R2LSH builds multiple query-centric balls in a group of two-dimensional projected subspaces, while VHP utilizes the information of one-dimensional distances between the query and data points on every projected vector. Both of these two methods can prune false points more accurately than the existing query-aware LSH variants.

In the second part, we focus on the inner product space and solve the maximum inner product search problem. For this purpose, we propose a query-aware LSH variant called AdaLSH which is built on a multi-ring structure. The basic idea of AdaLSH is that, based on different norms of data points, we can control adaptively the width of the query-aware search windows such that those points having larger norms can be examined more carefully to ensure the accuracy. In order to realize this idea, we design a multi-round search strategy such that query-aware search windows in different rings extend at different paces, which not only achieves the goal mentioned above, but also significantly decreases the total searching cost.

Since all three proposed methods are based on LSH, all of the proposed algorithms possess probability guarantees in accuracy. Extensive experiments confirm the superiority of these methods over existing state-of-the-art methods. In particular, R2LSH and VHP scale up to billion-scale datasets, because they are disk-based.

Acknowledgements

First, I would like to express the deepest gratitude to my supervisor, Professor Mineichi Kudo, for his help in various aspects in the past four years. He offered me the chance to study and conduct the research in Hokkaido University, and gave me the freedom to the largest extent to research the problem of my interest. For my each paper, he examined it word by word and gave me many constructive suggestions to improve its quality. I will always keep his help and rigorous attitude to the research in mind.

I would also like to thank Associate Professor Atsuyoshi Nakamura for his invaluable suggestions for my future plan. My sincere thanks also go to my research committee members, Professor Hideyuki Imai and Professor Akira Tanaka, for offering constructive comments on my research work. I also thank all the members in the PRML lab and really enjoy the experience in the lab in the past three years. Especially, I thank the root members in PRML lab for their arduous work in managing servers to ensure that I could conduct my experiments smoothly.

Finally, I would like to thank my parents. Without their financial support and encouragement, I could not make any progress in the research filed. Also, I would like to thank my former supervisor Professor Hongya Wang and Assistant Professor Lu Sun for their help and suggestions before and after I started my study in Hokkaido University.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Background

With the arrival of big data era, similarity search plays an important role in many fields such as multimedia database, signal processing, space-time database, data mining and analysis system[4, 32]. Especially after 1970s, similarity search is found in applications such as computer aided design (CAD), geographic information system (GIS) and bioinformatics. Faced with million-scale datasets, even billion-scale datasets, the goal of similarity search is to find the "closest" objects of issued queries efficiently and accurately. For different applications, the tackled data types, data spaces and similarity measures vary much. For example, in the field of text mining, the widely used measure is Jaccard similarity while in the filed of image retrieval, the widely used measures are ℓ_2 metric and cosine metric. In this paper, we only focus on the latter situation. More precisely, we deal with those datasets whose objects are represented by high-dimensional vectors in a fixed Euclidean space.

From the goal of similarity search introduced in the preceding paragraph, it is easy to see that such problems can always be solved by brute force, or say, the linear scan. That is, we scan all objects in the dataset sequentially and maintain the closest one during the examination process. However, such solution is unacceptable for most of real applications due to the following two reasons. (1) The tackled real datasets are both large-scale and high-dimensional. This means that the time complexity $O(nd)$ for a single query is too high to be accepted in practice, where n and d are the data size and the data dimension, respectively. (2) For real-time systems, we have to process a bunch of queries issued by users rather than a single query during a period of time.

In this situation, the high time cost of linear scan will accumulate and lead to a very long response time. Therefore, it is timely and important to design efficient methods to solve similarity search problems. It is notable that currently, exact nearest neighbor search problem (ENN), that is, finding the true nearest neighbor of the query, is often thought to be intractable in the sub-linear time complexity. Thus, most of researchers focus on its weaker version and only try to find approximate nearest neighbors (ANN), that is, those points whose distances to the query are close to the smallest distance, since finding such points are also acceptable for many applications in the real world.

Before proceeding, we need to point out that, even for the weak version, similarity search problems are still non-trivial in high-dimensional spaces since, as the dimension increases, many data points have very close distances to the query such that they are approximately distributed on a query-centric sphere from the geometric viewpoint, in which case, the true nearest neighbors of queries are hard to be distinguished from other points. A straightforward consequence is that, the performances of those well-known tree-based data structures, such as, *R-tree* [23], *kd-tree* [10], *sr-tree* [31] degrade significantly and run even slower than the linear scan in some cases [55]. This phenomenon is called *the curse of dimensionality* [28].

It is widely accepted that an efficient solution to overcome this difficulty is to resort to the projection techniques, which are based on the following intuitive fact: two close points in the original space is very likely to be also close in the projected spaces. By processing points in such low-dimensional spaces, we can lower the time complexity of linear scan. Based on this fact, various projection-based methods have been proposed and they can be roughly divided into the following two categories: (1) reducing the data dimension directly based on the data distribution, and (2) building hash functions. For methods in the first category, relatively less important subspace of the original space will be removed. Then some search algorithms will be implemented in the subspace with the reduced dimension. Representatives of such type are principal component analysis (PCA) [24], linear discriminant analysis (LDA), manifold learning algorithms such as LLE, Laplacian Eigenmap [8] [9]. The methods in the second category can be further divided into two sub-categories: (1) building data-dependent hash functions, and (2) building data-independent hash functions. Here, data-dependent hash functions refer to those functions which are learned by utilizing characteristics or distributions

of datasets. They can be further divided into unsupervised hash functions and supervised hash functions. Representatives of unsupervised hash functions are KLSH [46], spectral hashing [56]. For data-independent hash functions, the construction of them is independent of the distribution of the dataset and only depends on the used similarity measure. As a typical example, *locality sensitive hashing (LSH)* is a data-independent projection technique and is widely adopted in various applications. It has been proved that LSH-based methods are superior to many other algorithms in terms of space consumption and query efficiency [2]. This paper will focus on the design of LSH-based indexes/methods to solve the similarity search problems.

1.2 Locality sensitive hashing

In this section, we briefly review the basic ideas of Locality Sensitive Hashing(LSH). First, we give the following definition of *r-nearest neighbor search problem*: for a given query q, find a point o whose distance to the query is not greater than r. To solve the r-near neighbor search problem, Indyk and Motwani introduced the concept of LSH in their influential paper [28]. The idea of random projection on which LSH based, however, can be traced back to much earlier work in [11, 33]. The rationale behind LSH is that, by using specific hashing functions, we can hash the points such that the probability of collision for data points which are close to each other is much higher than that for those which are far apart. In this section, we use $\mathcal{H}$ to denote a family of hash functions mapping $\mathbb{R}^d$ to some universe $\mathcal{U}$. For any two points o and q and a hash function h that is chosen from $\mathcal{H}$ uniformly at random, the family $\mathcal{H}$ is called locality sensitive if the probability that these two points collide ($h(q) = h(o)$) satisfies the following condition, where $c > 1$ is a user-specified parameter called *approximation ratio*. For an LSH family to be useful, it has to satisfy $p^1 > p^2$.

Definition 1 ([28]). *A family $\mathcal{H}$ of hash functions is called (r, cr, p^1, p^2)-sensitive if for any two points o, $q \in \mathbb{R}^d$.*

- *if $\|q - o\| \leq r$ then $Pr_{\mathcal{H}}[h(q) = h(o)] \geq p^1$*
- *if $\|q - o\| \geq cr$ then $Pr_{\mathcal{H}}[h(q) = h(o)] \leq p^2$*

This condition can be applied to various measures. In the following introduction, we focus on the LSH family designed for the Euclidian space. In [17], the authors proposed the following LSH family for ℓ_2 metric: pick a group of random vectors $\vec{a}'s$ in $\mathbb{R}^n$ and project o onto $\vec{a}'s$. The 1-dimensional line ($\vec{a}$) is then chopped into segments of length w. The number of segment which o falls into, after shifted by a random value $b \in [0, w)$, is the hash value of o. Formally, $h_{a,b}(o) = \left\lfloor \frac{\vec{a}\cdot\vec{o}+b}{w} \right\rfloor$. Each coordinate of $\vec{a}$ is drawn uniformly at random following the Gaussian distribution. The probability of collision that any two points at distance r collides over the random vector $\vec{a}$ is as follows:

$$p^e_{\mathbf{h}}(r) = \int_0^w \frac{1}{r} g(\frac{t}{r})(1 - \frac{t}{w})dt \tag{1.1}$$

where $g(x) = 2f(x)$ and $f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$.

The LSH approach to the approximate NNS problem is based on the existence of the locality sensitive hash functions. Given a family of LSH functions $\mathcal{H}$, the classic LSH method works as follows [17, 28]. For parameters k and ℓ, ℓ functions $g_j(q) = (h_{1,j}(q), \ldots, h_{k,j}(q))$ are chosen, where $h_{i,j}(1 \le i \le k, 1 \le j \le \ell)$ is drawn independently and uniformly at random from $\mathcal{H}$ [28]. Given a dataset $\mathcal{D}$, $\forall o \in \mathcal{D}$ the bucket $g_j(o)$ is computed for $j = 1, \ldots, \ell$, and then o is inserted into the corresponding bucket. To process a query q, one has to compute $g_j(q), j = 1, \ldots, \ell$, first, and then retrieve all points that lie in at least one of these buckets. For simplicity, we denote $\mathbf{h}$ and $\mathbf{h}^\ell_k$ as a random LSH function and a random LSH data structure respectively, and the (uniformly drawn) samples of $\mathbf{h}^\ell_k$ and $\mathbf{h}$ are denoted by h^ℓ_k and h, respectively.

As mentioned in Section 1.1, LSH and its variants are capable of providing, with some constant success probability, excellent asymptotic performance in terms of space consumption and query cost [20, 28, 40, 52]. The theoretical guarantee relies on the fact that, for any point pair $\langle q, o\rangle$ such that $d(q, o) = r$, where $d(q, o)$ is the distance between q and o, the collision probability $p_{\mathbf{h}}(r)$ that $\langle q, o\rangle$ collides over $\mathbf{h}$ decreases monotonically with r [28]. As a quick result, the collision probability of $\langle q, o\rangle$ over $\mathbf{h}^\ell_k$, denoted by $P_{\mathbf{h}^\ell_k}(r)$, can be calculated using Equation (1.2) since $k \times \ell$ hash functions are drawn independently from $\mathcal{H}$.

$$P_{\mathbf{h}^\ell_k}(r) = 1 - (1 - p_{\mathbf{h}}(r)^k)^\ell \tag{1.2}$$

Table 1.1: Publication list

Method	Conference/Journal	Solved problem	Section
VHP [39]	PVLDB'20	ANNS	Sec. 2.4
R2LSH [37]	ICDE'20	ANNS	Sec. 2.5
AdaLSH [38]	IEICE'21	MIPS	Sec. 3

From this expression, it is easy to see that, by reasonably choosing the values of k and ℓ, LSH could ensure that the difference between $P_{\mathbf{h}_k^\ell}(r)$ and $P_{\mathbf{h}_k^\ell}(cr)$ is large enough and thus distinguish those points within the query-centric sphere of radius r and points outside the query-centric sphere of radius cr efficiently. In the later chapters, we will see that this property is the starting point for almost all LSH-based methods.

1.3 Organization of this paper

The relevant published or accepted papers are listed in Table 1.1 and the rest of this paper is organized as follows. In Chapter 2, we will introduce the applications of LSH in the Euclidean space with ℓ_2 metric, and propose two LSH-based methods VHP and R2LSH. Specifically, VHP will be introduced in Section 2.4.2 and R2LSH will be introduced in Section 2.5. In Chapter 3, we will introduce the applications of LSH in the inner product space and propose a related method called Adaptive LSH. In Chapter 4, we will conclude this paper and provide some ideas on the the future research.

Chapter 2

Approximate Nearest Neighbor Search Problem

2.1 Introduction

In Chapter 2, we focus on *the approximate nearest neighbor search problem (ANNS)* in a high-dimensional Euclidean space with ℓ_2 metric. To remove the curse of dimensionality mentioned in Chapter 1, the common wisdom is to design efficient c-approximate NN search algorithms by trading precision for speed [12]. A point o is called a *c-approximate NN* (c-ANN) of query q if its distance to q is at most c times the distance from q to its exact NN o_*, i.e., $\|q, o\| \leq c\|q, o_*\|$, where c is the *approximation ratio*. As one of the most promising c-ANN search algorithms, Locality Sensitive Hashing (LSH) owns attractive query performance and probability guarantee in theory [28], and finds broad applications in practice [2, 32].

As a matter of fact, the original LSH method (E2LSH) whose implementation is introduced in Section 1.2, does not support c-ANN search directly and a naive extension may cause prohibitively large storage cost [52]. To this end, several LSH variants such as LSH-forest [52], C2LSH [20] and QALSH [26], have been proposed in order to answer c-ANN queries with reasonably small indexes, constant success probability and sub-linear query overhead. Inspired by these works, we proposed two novel and efficient LSH-based methods called VHP and R2LSH which could improve the performances of existing methods further. Both proposed methods are disk-based and their details will be introduced from Section 2.4.

2.2 Related work

Before proceeding, let us review some important and efficient ANNS methods. Although this paper mainly focuses on the design of LSH-based methods, some methods of other types will also be mentioned in Sec 2.2.2 for the completeness.

2.2.1 LSH-based Algorithms

Among the approximate NN search algorithms, the Locality Sensitive Hashing is the most widely used one due to its excellent theoretical guarantees and empirical performance. E2LSH, the classical LSH implementations for ℓ_2 norm, cannot solve c-ANN search problem directly. In practice, one has to either assume there exists a "magical" radius r, which can lead arbitrarily bad outputs, or uses multiple hashing tables tailored for different raddi, which may lead to prohibitively large space consumption in indexing. To reduce the storage cost, LSB-Forest [52] and C2LSH [20] use the so-called virtual rehashing technique, implicitly or explicitly, to avoid building physical hash tables for each search radius. The index size of LSB-Forest is far greater than that of C2LSH because the former ensures that the worst-case I/O cost is sub-linear to both data size n and data dimension d whereas the latter has no such guarantee - it only bounds the number of candidates by some constant but ignores the overhead in index access.

Based on the idea of query-aware hashing, the two state-of-the-art algorithms QALSH and SRS further improve the efficiency over C2LSH by using different index structures and search methods, respectively. SRS uses an m-dimensional R-tree (typically $m \leq 10$) to store the $\langle g(o), o_{id} \rangle$ pair for each point o and transforms the c-ANN search in the d-dimensional space into the range query in the m-dimensional projection space. The rationale is that the probability that a point o is the NN of q decreases as $\Delta_m(o)$ increases, where $\Delta_m(o) = \|g_m(q) - g_m(o)\|$ During c-ANN search, points are accessed according to the increasing order of their $\Delta_m(o)$. QALSH uses the so-called *dynamic collision counting* technique to identify eligible candidates. Briefly, one hash function $h(\cdot)$ defines many buckets (search windows) and two points collide if they fall into the same bucket. With a compound hash function $g_m(\cdot) = \langle h_1(\cdot), h_2(\cdot), \cdots, h_m(\cdot) \rangle$, o is mapped from the feature space into the m-dimensional projection space. Point o collides with q over $g_m(\cdot)$ if the collision number out of m hash functions is no less

than L, where L is a pre-defined collision threshold. All points that collide with q are regarded as candidates and further screened by QALSH.

Motivated by the observation that the optimal ℓ_p metric is application-dependent, LazyLSH [57] is proposed to solve the NN search problem for the fractional distance metrics, i.e., ℓ_p metrics ($0 < p < 1$) with a single index. FALCONN is the state-of-the-art LSH scheme for the angular distance, both theoretically and practically [3]. Except for E2LSH and FALCONN, the other algorithms are disk-based and thus can handle datasets that do not fit into the memory.

All of the aforementioned LSH algorithms provide probability guarantees on the result quality (recall and/or precision). To achieve better efficiency, many LSH extensions such as Multi-probe LSH [40], SK-LSH [36], LSH-forest [7] and Selective hashing [22] use heuristics to access more plausible buckets or re-organize datasets, and do not ensure any LSH-like theoretical guarantee.

2.2.2 Non-LSH Algorithms

Inspired by LSH, a vast amount of research efforts have been devoted to the learning-based hashing for ANN search. Spectral hashing utilizes the spectral graph analysis technique to embed points to the Hamming space based on the pairwise similarity matrix [56]. DSH learns the LSH functions for kNN search directly by computing the minimal general eigenvector and then optimizing the hash functions iteratively with the boosting technique [21]. Production quantization (PQ) divides the feature space into disjoint subspaces and then quantizes each subspace separately into multiple clusters [29]. By concatenating codes from different subspaces together, PQ partitions the feature space into a large number of fine-grained clusters which enables efficient NN search. As pointed in [54], the high training cost (preprocessing overhead) is a challenging problem for learning to hash while dealing with large datasets. Moreover, almost all learning-based hashing methods are memory-based and do not ensure the answer quality theoretically.

FLANN [42] is a meta algorithm which selects the most suitable techniques among randomized kd-tree, hierarchical k-mean tree and linear scan for a specific dataset. As a representative of graph-based algorithms, HNSW uses long-range links to simulate the small-world property based on an approximation of the Delaunay graph [41]. The experiment study in a recent paper shows that the main-memory-based ANN algorithms

such as HNSW and PQ find difficulty to work with large datasets in a commodity PC [5]. HD-Index [5] is proposed to support the approximate NN search for disk-based billion-scale datasets. HD-Index consists of a set of hierarchical structures called RDB-trees built on Hilbert keys of database objects. In the query phase, those objects which are close to the query in an arbitrary RDBtree are determined as candidates. Again, the best-effort nature makes these tree-based and graph-based algorithms devoid of theoretical guarantee.

It is notable that Ciaccia and Patella have also considered using a hypersphere to delimit the search space and proposed an algorithm PAC-NN to support probabilistic kANN queries [14]. A recent experimental study extends some exact NN algorithms, i.e., iSAX2+ and DSTree, to support PAC (probably approximately correct) NN query [19]. The extension is based on the idea of PAC-NN [14], which makes the probabilistic iSAX2+ and DSTree data dependent. As a result, it is reported that they demonstrate better accuracy and efficiency than SRS and QALSH.

2.3 Preliminary

In Chapter 2, we focus on the Euclidean space with ℓ_2 norm. For a dataset $\mathcal{D}$ of N d-dimensional data points, NN search finds the point o_* in $\mathcal{D}$ with the minimum distance to query q. For c-ANN search, only a c-approximate neighbor o needs to be returned, that is, $\|q-o\| \leq c\,\|q-o_*\|$, where $\|q-o\|$ denotes the ℓ_2 distance between q and o. kNN search returns k results o_{*_j} $(1 \leq j \leq k)$, where o_{*_j} is the j-th nearest neighbor of q. Its c-approximate version, c-k-ANN, returns a set of k objects o_j $(1 \leq j \leq k)$ satisfying $\|q-o_j\| \leq c\,\|q-o_{*_j}\|$.

Let $d(o_1, o_2)$ denote $\|o_1 - o_2\|$. Suppose $\vec{a} = [a_1, a_2, \cdots, a_m]$ is a random projection vector, each entry of which is an i.i.d. random variable following the standard normal distribution $\mathcal{N}(0,1)$. The inner product between $\vec{a}$ and vector $\vec{o}$, denoted as $h(o) = \langle \vec{a}, \vec{o} \rangle$ is an LSH signature of o. We have the following important Lemma [17].

Lemma 1. *For any points o_1, o_2 in $\mathbb{R}^d$, $h(o_1) - h(o_2)$ follows the normal distribution $\mathcal{N}(0, d^2(o_1, o_2))$.*

Lemma 1 holds due to the fact that the standard normal distribution $\mathcal{N}(0,1)$ is a p-stable distribution for $p = 2$. Lemma 1 suggests that the difference between two LSH signatures follows the normal distribution with mean 0 and standard deviation

$d(o_1, o_2)$, i.e., the Euclidian distance between the two original points. This establishes analytical connection between the distances in the projection space and original feature space, which is the building block for constructing the LSH family.

Given a positive real number t, the interval $[h(q) - t,\ h(q) + t]$ is referred to as a *query-aware search window*. For ease of presentation, we will refer to it as a search window $[-t, t]$ henceforth. For any point o, the probability $p(s)$ that $h(o)$ $(s = d(o, q))$ falls into this window is given by Equation (2.1) [26].

$$p(s) = Pr[\delta(o) \leq t] = \int_{-\frac{t}{s}}^{\frac{t}{s}} \varphi(x)dx, \tag{2.1}$$

where $\delta(o) = |h(q) - h(o)|$ and $\varphi(x)$ is the probability density function (PDF) of $\mathcal{N}(0, 1)$.

It is easy to see that, for fixed t, $p(s)$ is a monotonically decreasing function, which means that the probability that q and o fall into the same search window decreases with their Euclidian distance. According to the definition of locality sensitive hashing, $h(\cdot)$ is shown to be the query-aware LSH family [26].

Suppose X follows the normal distribution $\mathcal{N}(\mu, \sigma^2)$ and lies within the interval $X \in [a, b],\ -\infty \leq a < b \leq \infty$. Then X conditional on $a \leq X \leq b$ has a truncated normal distribution $\mathcal{N}(x \in [a, b]; \mu, \sigma^2)$. Its probability density function, f, in support $[a, b]$, is defined by:

$$f(x; \mu, \sigma^2, a, b) = \frac{\varphi(\frac{x-\mu}{\sigma})}{\Phi(\frac{b-\mu}{\sigma}) - \Phi(\frac{a-\mu}{\sigma})} \tag{2.2}$$

where $\Phi(x)$ is the CDF of the standard normal distribution.

In fact, truncated normal distribution is important for the analysis of VHP introduced later since, instead of only caring about whether q and o fall into the same bucket, we exploit the precise position information of o to obtain a more fine-grained filtering condition. Suppose o lies in the query-aware search window already, then $h(q) - h(o)$ follows the truncated normal distribution $f(x; 0, d^2(o_1, o_2), -t, t)$ instead of the normal distribution $\mathcal{N}(0, d^2(o_1, o_2))$.

Next, we start from Equation (2.1) in another direction. Let $g(o) = (h_1(o), h_2(o), \ldots, h_m(o))$ denote the random mapping of o from the *original* d-dimensional space to the m-dimensional *projected space* and $\Delta(o) = \|g(q) - g(o)\|$. Sun et al. [51] observed that $\Delta^2(o)/\|q - o\|^2$ follows the $\chi^2(m)$ distribution. Thus, as a generalization of Lemma 1,

the probability $p(s,r)$ that $g(o)$ ($s = \|q-o\|$) falls within the hyperball of radius r centered at $g(q)$ is expressed as

$$p(s,r) = \Pr[\Delta(o) \leq r] = \Psi_m(\frac{r^2}{\|q-o\|^2}), \tag{2.3}$$

where $\Psi_m(x)$ is the cumulative distribution function (CDF) of the $\chi^2(m)$ distribution. Equation (2.3) will be the building block for the theoretical analysis of R2LSH.

2.4 Virtual Hypersphere Partitioning

2.4.1 Motivation

In this section, we discuss how QALSH works and its limitations from a geometric point of view. Compared with C2LSH, QALSH applies the query-aware search window $[-t,t]$ to each hash function $h_i(\cdot)$. Point o collides with q with respect to $h_i(\cdot)$ if $-t \leq h_i(o) - h_i(q) \leq t$.

Given query q and the search window of size $2t$, point o might not collide with q over all m random hash functions. To distinguish relevant and irrelevant points, QALSH counts the collision number for each point. If the collision number is greater than some given threshold L, it is said that o collides with q over $g_m(\cdot)$. A formal treatment for this is given in inequality (2.4).

$$|\{i, 1 \leq i \leq m \mid |h_i(o) - h_i(q)| \leq t\}| \geq L \tag{2.4}$$

In QALSH, the exact ℓ_2 distance between o and q is evaluated only if o collides with q over $g_m(\cdot)$, which avoids the traversal of whole dataset $\mathcal{D}$. In the quick example in Figure 2.1, three hash functions are used and the search window size is 9 (t=4.5). Suppose the counting threshold $L = 2$, QALSH will mark o_2 and o_3 as relevant points because they appear in the search windows twice and leave o_1 untouched.

In this subsection, we will examine the principle of QALSH from a geometric point of view, whereby its limitations are outlined. For the statement that o collides with q w.r.t. $h(\cdot)$, a geometric interpretation is that o lies in the region bounded by two hyperplanes, defined by $\sum_{i=1}^{d} a_i x_i = h(q) - t$ and $\sum_{i=1}^{d} a_i x_i = h(q) + t$, in the d-dimensional space.

Similarly, for QALSH, visiting candidates (points which collide with q over $g_m(\cdot)$) is like checking points in the region bounded by $j \geq L$ hyperplane pairs, which are

Table 2.1: Notations

Notation	Explanation
$\varphi(x)$	the probability density function (PDF) of $\mathcal{N}(0,1)$.
$\Phi(x)$	the cumulative distribution function (CDF) of $\mathcal{N}(0,1)$.
P_*	the success probability specified by users.
L	the collision threshold used by QALSH.
o_*	the nearest neighbor of q.
o_{min}	the nearest neighbor returned by the NN search algorithm.
$d(o_1, o_2)$	the exact ℓ_2 distance between o_1 and o_2.
s_*	$s_* = d(o_*, q)$
m	the number of projection vectors.
$h(\cdot)$	the locality sensitive hash function.
$\delta_i(o)$	the ℓ_2 distance between $h_i(o)$ and $h_i(q)$.
$g_m(\cdot)$	the compound hash function $\langle\ h_1(\cdot), h_2(\cdot), \cdots, h_m(\cdot)\ \rangle$.
l_i	the radius of physical hypersphere in the i-constrained projection subspace.
$\tilde{\sigma}(l_i)$	the radius of the virtual hypersphere associated with i and l_i.
$[-t, t]$	the search window of size $2t$.
$\mathfrak{I}_t(o)$	The set of hash functions satisfying $\lvert h_i(q) - h_i(o)\rvert \leq t$.
$r_t(o)$	the collision number of point o with respect to $[-t, t]$.
$\Delta^t(o)$	the observable projection distance of point o in the $r_t(o)$-constrained projection subspace.
$\tilde{d}(o, q)$	$\tilde{d}(o, q) = \tilde{\sigma}(\Delta^t(o))$ is the estimated distance from o to q.

defined by j different hash functions. Since m, the number of projections, is often less than the dimensionality d of the ambient space, the search space is actually irregular and unbounded. It is difficult to visualize such space in high-dimensional cases, and thus we depict a simple example in 2-dimensional space to train the reader's intuition.

As illustrated in Figure 2.2, the dimensionality d of the feature space is 2, the number of hash functions m is 2 and the collision threshold L is set to 1. The search window is of size $2t$ and the solid line represents a degraded hyperplane in the 2-dimensional space. The crossroad-like region is the search space of collision-threshold-based filtering, which is irregular and unbounded. Points in this region are estimated to be the NNs by QALSH. In contrast, virtual sphere partitioning imposes a virtual hypersphere in the feature space, which is isotropic and bounded. Points whose distances to q are less than the radius of the hypersphere (in estimation) are regarded as candidates.

From this examples, it is easy to see that the search strategy of QALSH has two

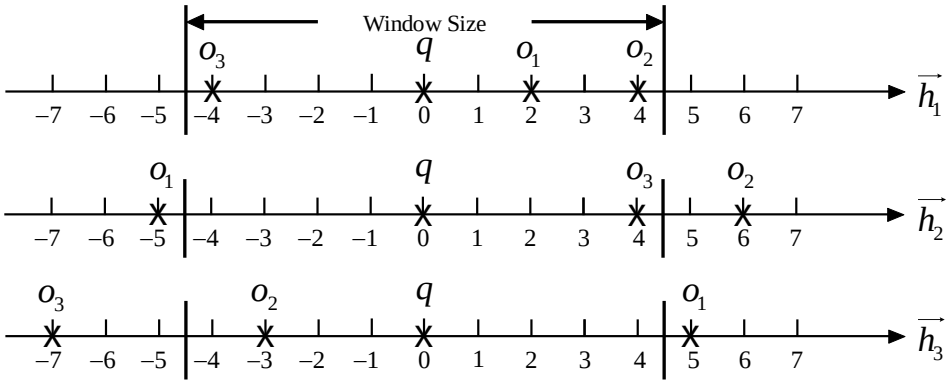


Figure 2.1: A running example of query-centric hashing.

limitations: (1) close points in the red areas (with collision number of 0) are missed and (2) many irrelevant points that are far away from q (outside the hypersphere but inside the crossroad-like region) may be examined since the region is unbounded. To remedy these limitations, we will propose a more fine-grained filtering strategy called VHP in the following discussion.

2.4.2 Virtual Hypersphere Partitioning

In this section, we present a novel disk-based indexing and searching algorithm VHP. The idea of virtual hypersphere partitioning and an illustrative example of query processing workflow are given in Section 2.4.2.1 and Section 2.4.2.2, respectively. The detailed algorithm is described in Section 2.4.2.3.

2.4.2.1 The Idea

In view of the limitations of QALSH discussed earlier, we suggest to use a hypersphere centered at the query, which is isotropic and bounded, to partition the original feature space and distinguish promising candidates and irrelevant ones. The idea is illustrated in Figure 2.2, where the inner region of the hypersphere is the search space. Since imposing a real hypersphere directly in the original space is difficult, we propose to use multiple physical hyperspheres to achieve the same goal. A few notations and definitions are needed before we present our proposal.

Recall that the compound hash function $g_m(\cdot)$ maps point o in $\mathbb{R}^d$ into the m-dimensional projection space $\mathbb{R}^m$. Due to the existence of search window $[-t, t]$, a

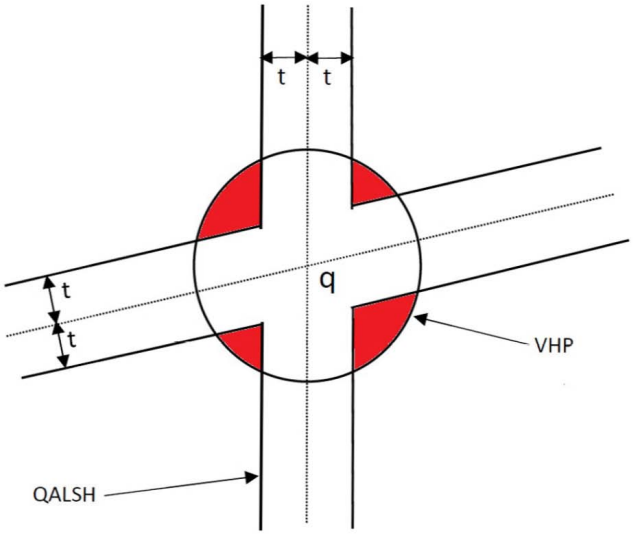


Figure 2.2: An illustrative example of the search spaces of virtual sphere partitioning and collision-threshold-based filtering.

point may lie in a query-centric i-constrained projection subspace, which is defined as follows.

Definition 2. *A query-centric i-constrained projection subspace is composed of $x \in \mathbb{R}^m$ such that $h_j(q) - t \leq x_j \leq h_j(q) + t$ $(1 \leq j \leq i)$ for any i out of m hash functions.*

Let $\mathcal{I}_t(o)$ denote the set of hash functions spanning the i-constrained projection subspace that o sits in and $r_t(o) = |\mathcal{I}_t(o)|$. We denote by $\Delta^t(o)$ the distance between q and o in this subspace. Take Figure 2.1 as an example, o_1 and o_2 lie in the 1-constrained and 2-constrained projection subspaces, respectively. For o_2, we have $\mathcal{I}_{4.5}(o_2) = \{h_1, h_3\}$ and $r_{4.5}(o_2) = 2$. $\Delta^{4.5}(o_2) = 5$ since $h_1(o_2) - h_1(q) = 4$ and $h_3(q) - h_3(o_2) = 3$, thus their Euclidian distance is $\sqrt{4^2 + 3^2}$ =5. In the sequel, we will omit the term i-constrained if it is obvious from the context.

There are m classes of the i-constrained $(1 \leq i \leq m)$ projection subspaces in total and m choose i i-constrained projection subspaces for each given i. Obviously, different points may lie in different projection subspaces. For point o in any one of the i-constrained projection subspaces, o is regarded as a candidate only if $\Delta^t(o) \leq l_i$, which is like imposing a physical hyperspheres of radius l_i, centered at the projection signature of q, to distinguish candidates and irrelevant points. As will be discussed in Section 2.4.2.4, such a physical hypersphere is equivalent (in estimation) to a virtual

hypersphere with radius $\tilde{\sigma}(l_i)$ in the original space. Moreover, checking points such that $\Delta^t(o) \leq l_i$ is like examining candidates satisfying $\tilde{d}(o,q) \leq \tilde{\sigma}(l_i)$.

We say that o collides with q under virtual hypersphere partitioning, i.e., o is a candidate, if $\Delta^t(o) \leq l_i$ for any $1 \leq i \leq m$, that is, Equation (2.5) holds. Note that the statement o lies in some i-constrained projection subspace is equivalent to o collides with q w.r.t. $g(\cdot)$ i times.

$$\bigvee_i \{r_t(o) = i \wedge \Delta^t(o) \leq l_i\}, 1 \leq i \leq m \tag{2.5}$$

It is easy to see that Equation (2.5) is more stringent and will degrade to Inequality (2.4) if one sets $l_i = 0$ for $1 \leq i < L$ and $l_i = +\infty$ for $L \leq i \leq m$.

m physical hyperspheres lead to m virtual hyperspheres in the original space, which may be of different radii. To emulate a single virtual hypersphere, we judiciously choose l_i to make the radii of the m virtual hyperspheres identical with each other. In this way, using Equation (2.5) as a filtering condition is like examining points whose exact distances to q (in estimation) are less than the virtual radius.

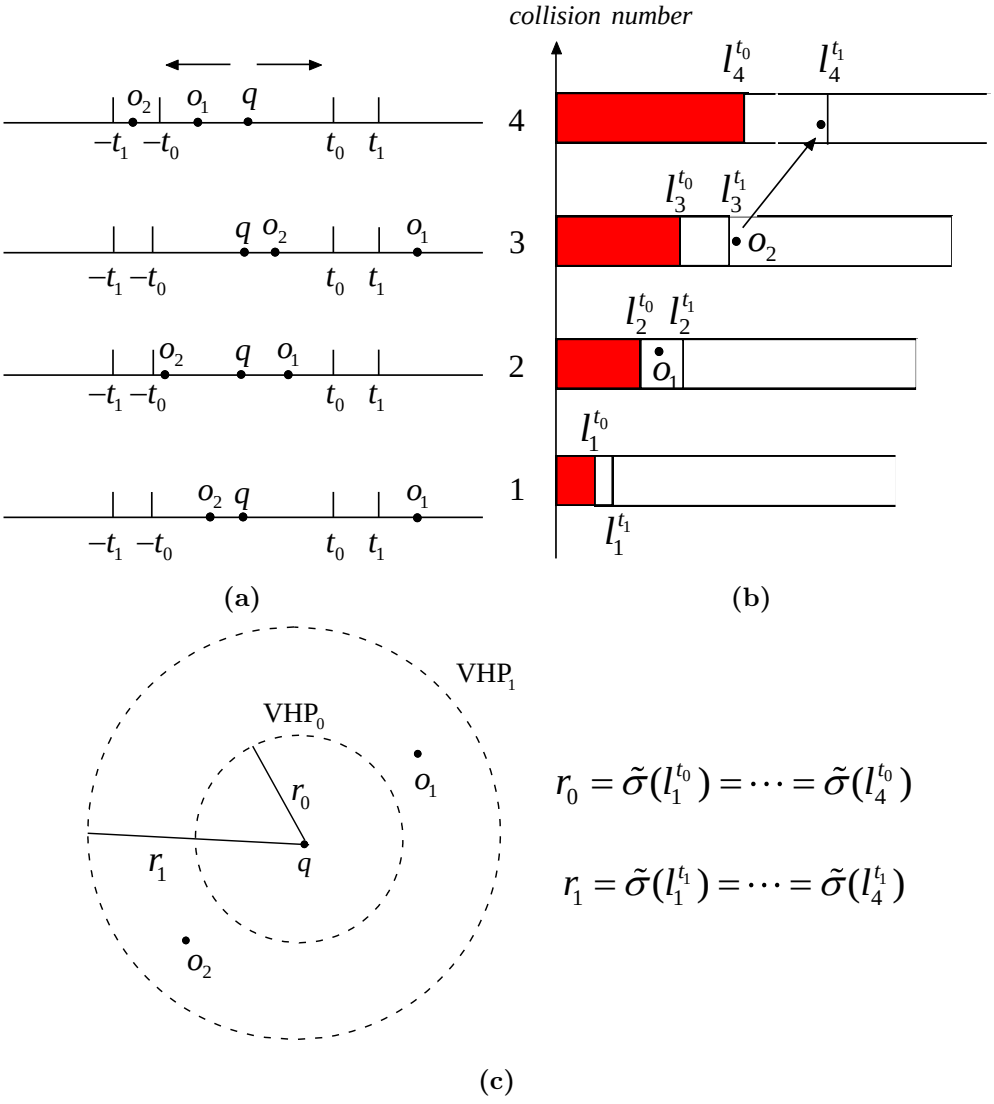
2.4.2.2 An Illustrative Example of Query Processing Workflow

In this subsection, we highlight the workflow of the proposed solution using an illustrative example as shown in Figure 2.3.

Before query processing, we need to set proper l_i to guarantee the result quality. As will be discussed in Section 2.4.2.4, the radii of physical hyperspheres depend on the distance between the given query and its NN. To circumvent this issue, we first calculate the *base distance thresholds* $l_i^{t_0}$ in an off-line fashion for user-specified success probability, under the assumption that the *base search window* is $[-t_0, t_0]$ and $d(o_*, q) = 1$[1].

As illustrated in Figure 2.3(a) and Figure 2.3(b), the half-width of search window and radii of physical hyperspheres are set to t_0 and $l_i^{t_0}$ $(1 \leq i \leq m)$ in the beginning. The corresponding virtual hypersphere VHP_0 is depicted in Figure 2.3(c). Please note that, while $l_i^{t_0}$ are of different values, they are chosen judiciously such that they are equivalent to the radius of VHP_0. When $t = t_0$, both o_1 and o_2 are not located in

[1]In practice, we may set $d(q, o_*)$ to the minimum possible NN distance. We set $d(q, o_*) = 1$ here for ease of presentation.



$$r_0 = \tilde{\sigma}(l_1^{t_0}) = \cdots = \tilde{\sigma}(l_4^{t_0})$$

$$r_1 = \tilde{\sigma}(l_1^{t_1}) = \cdots = \tilde{\sigma}(l_4^{t_1})$$

Figure 2.3: An illustrative example of how VHP works.

VHP_0 because their estimated distances to q in the feature space are greater than the corresponding hypersphere radius, that is, $\tilde{d}(o_1, q) > r_0$ and $\tilde{d}(o_2, q) > r_0$. This is computationally done by evaluating $\Delta^{t_0}(o_2) > l_3^{t_0}$ and $\Delta^{t_0}(o_1) > l_2^{t_0}$ in the respective projection subspaces (Figure 2.3(b)).

Figure 2.3 also illustrates how the search window, radii of physical hyperspheres and the virtual hypersphere grow coordinately. To accommodate more candidates, VHP extends the search window from t_0 to t_1 first (Figure 2.3(a)). As a result, o_2 jumps from 3-constrained to 4-constrained projection subspace while o_1 keeps the same collision number with q. The physical hypersphere radii are updated from $\{l_i^{t_0}\}$ to $\{l_i^{t_1}\}$ accordingly as shown in Figure 2.3(b). As one can see, both o_1 and o_2 are identified as candidates since $\Delta^{t_1}(o_2) \leq l_4^{t_1}$ and $\Delta^{t_1}(o_1) \leq l_2^{t_1}$. The equivalent effect is illustrated in Figure 2.3(c), where o_1 and o_2 are bounded by the enlarged virtual hypersphere VHP_1. Please note that the radii $\tilde{\sigma}(l_i)$ of all virtual hyperspheres are identical with each other all the time.

By extending the search window and hypersphere radii gradually, VHP is able to find o_* no matter how far it is away from q. The theoretical analysis in Section 2.4.2.5 and Section 2.4.3.1 guarantees that, for arbitrary $d(o_*, q)$, o_* will be found with probability at least P_* when the search window extends to $[-d(o_{min}, q)t_0, d(o_{min}, q)t_0]$ and the radii reach $d(o_{min}, q)l_i^{t_0}$, where o_{min} is the nearest point found by VHP so far.

2.4.2.3 The Algorithm

Index Building Phase: To index the data, m LSH random projections $\vec{a}_i$ are generated first. Then, each $o \in \mathcal{D}$ is projected from the d-dimensional feature space into m 1-dimensional spaces. For each projection vector $\vec{a}_i$, a sorted list is built to store the hash values and object identifiers for all points, and the list is sorted in the ascending order of $h_i(o)$. Finally, we index each sorted list using a B^+-tree and store it on the disk.

NN Search Phase: When a query q arrives, we perform a range search $[h(q) - t, h(q) + t]$ over each B^+-tree for given search window of size $2t$. During the range search, each point o is associated with 2-tuple $\langle r_t(o), \Delta^t(o)\rangle$. Recall that $r_t(o)$ denotes the collision number and $\Delta^t(o)$ refers to the distance between o and q in the $r_t(o)$-constrained projection subspace. Take o_2 in Figure 2.1 as an example, $r_{4.5}(o_2) = 2$ and $\Delta^{4.5}(o_2) = 5$.

Algorithm 1: VHP(q; c, t_0, $(l_1^{t_0}, l_2^{t_0}, ..., l_m^{t_0})$)

Input: q is the query point; $c(c \geq 1)$ is the approximation ratio; $2t_0$ and $(l_1^{t_0}, l_2^{t_0}, ..., l_m^{t_0})$ are the base search window size and base radii, respectively;

Output: $o_{\min}$

1 $t = 0$; $o_{\min}$ = a point at infinity;
2 **while** $d(o_{\min}, q)/c > \frac{t}{t_0}$ **do**
3 $t = t + \Delta t$ $(\Delta t > 0)$;
4 $\forall o \in \mathcal{D}$ update $r_t(o)$ and $\Delta^t(o)$ if necessary;
5 **if** *o is not visited and* $\Delta^t(o) \leq \frac{t}{t_0} l^{t_0}_{r_t(o)}$ **then**
6 calculate $d(o, q)$;
7 update $o_{\min}$ if necessary;
8 return $o_{\min}$

We present the probabilistic NN version of VHP in Algorithm 1, while leaving the c-k-ANN version to Section 2.4.3.2. It takes the query q as the input, as well as a set of parameters: the base search window of size $2t_0$ and the base hypersphere radii $(l_1^{t_0}, l_2^{t_0}, ..., l_m^{t_0})$. The parameters m, t_0 and $(l_1^{t_0}, l_2^{t_0}, ..., l_m^{t_0})$ are determined before the query processing. VHP returns the point $o_{\min}$ as the final answer.

Starting with t_0, VHP extends the search window gradually, which brings in more points. In each iteration, the 2-tuple $\langle r_t(o), \Delta^t(o) \rangle$ is updated if $r_t(o)$ increases (Line 4). The exact distance between q and o will be computed if $\Delta^t(o)$ is no greater than the radius $l^t_{r_t(o)} = \frac{t}{t_0} l^{t_0}_{r_t(o)}$. Then o_{min} is updated if necessary (Lines 5-7). The while loop terminates if the window size becomes large enough to meet the success probability (Line 2) and $o_{\min}$ is returned as the final answer (Line 8).

Update of windows size: Since VHP uses B^+-trees as the underlying index structure, there is a natural way to determine Δt (line 3 in Algorithm 1) as follows. We maintain a minimum heap of size $2m$, each element of which keeps track of the search direction (left or right) and offsets w.r.t. the query for a hash function. The increment in t (Δt) is determined in a data-driven fashion, i.e., VHP searches all B^+-trees until a new point is found in any B^+-tree and the position of this point determines the new window size.

2.4.2.4 Determine the Radii of Physical Hyperspheres

In this section, we will discuss how to determine the radii of physical hyperspheres, which is divided into the following three steps. (1) Step 1: we derive the collision probability between two points. (2) Step 2: we design a method to estimate the virtual radius for one physical hypersphere (3) Step 3: we prove the soundness of virtual hypersphere partitioning. The way to calculate the base hypersphere radii and the practical termination condition are presented in Section 2.4.2.5.

(1) **Collision Probability**. To conduct theoretical analysis for virtual hypersphere partitioning, we need to derive the collision probability for any two points first. To start with, some prerequisites are needed. Let $X_1, X_2, ..., X_j \in [-t, t]$ be i.i.d. random variables following the truncated normal distribution $\mathcal{N}(x \in [-t, t]; \mu, \sigma^2)$. Let $Y = \sqrt{\sum_{j=1}^{i} X_j^2}$ and obviously $Y \in [0, \sqrt{i}t]$. We use $\Omega_i^t(\mu, \sigma^2)$ to denote the distribution of Y and denote its CDF as $\mathcal{G}_i^t(x; \mu, \sigma^2)$.

Assume $d(o, q) = s$. Recall that $\delta(o) = h(q) - h(o)$ follows the normal distribution $\mathcal{N}(0, s^2)$ and $\mathcal{I}_t(o)$ denotes the set of $h(\cdot)$ over which o collides with q. We have the following important fact.

Fact 1. *For any $h_i(\cdot) \in \mathcal{I}_t(o)$, $\delta_i(o)$ follows the truncated normal distribution $\mathcal{N}(x \in [-t, t]; 0, s^2)$ and $\Delta^t(o)$ follows the distribution $\Omega_{r_t(o)}^t(0, s^2)$.*

Let A denote the event $\Delta^t(o) \leq l_{r_t(o)}$ and B denote the event $r_t(o) = i$ $(1 \leq i \leq m)$, thus the conditional probability $Pr[A|B]$ is:

$$Pr[A|B] = \mathcal{G}_i^t(l_i; 0, s^2)$$

It is easy to see that $r_t(o)$ obeys the Binomial distribution $\mathcal{B}(m, p(s))$, that is, $Pr[B] = C(m, i)(p(s))^i(1 - p(s))^{m-i}$. Then the joint probability $Pr[A \cap B]$ can be written as

$$Pr[A \cap B] = C(m, i)(p(s))^i(1 - p(s))^{m-i} \cdot \mathcal{G}_i^t(l_i; 0, s^2)$$

Since there are m classes of projection subspaces, the collision probability, denoted by $p_{\mathcal{L}}^t(s)$, can be calculated as follows, where $\mathcal{L} = (l_1, l_2, \cdots, l_m)$ is the set of radii of m hyperspheres.

$$p_{\mathcal{L}}^t(s) = \sum_{i=1}^{m} C(m, i)(p(s))^i(1 - p(s))^{m-i} \cdot \mathcal{G}_i^t(l_i; 0, s^2) \tag{2.6}$$

Suppose that we could know s_* beforehand. Then, to achieve the success probability P_*, we only need to choose proper m, t and $\mathcal{L}$ such that:

$$p_{\mathcal{L}}^{t}(s_*) = P_* \tag{2.7}$$

There may exist many $\mathcal{L}$'s that make Equation (2.7) hold. Next, we will show how to determine a unique and reasonable sequence $(l_1, l_2, ..., l_m)$ in order to fulfill virtual hypersphere partitioning.

(2) **Estimate the Virtual Radius for One Physical Hypersphere.** In this subsection, we focus on working out the estimate of the radius for one physical hypersphere in the feature space. To begin with, we need some notations and definitions first. An observation x sampled from the normal distribution $\mathcal{N}(0, \sigma^2)$ is called a *full observation* if it lies in the interval $t_1 \leq x \leq t_2$ and a *censored observation* otherwise, where t_1 and t_2 are two censoring points [15]. Here the term "censored" means that, instead of the exact value of this sample, we only know that it is situated outside the interval defined by the censoring points.

Suppose m i.i.d. samples $x_j, 1 \leq j \leq m$ are drawn from $\mathcal{N}(0, \sigma^2)$. Without the loss of generality, assume the first i samples are full observations, and there are c_1 censored observations such that $x < t_1$ and c_2 censored observations such that $x > t_2$. It is easy to see that $i + c_1 + c_2 = m$. Based on these evidences, the likelihood function $L(\sigma)$ is introduced in [15] to estimate the standard deviation of $\mathcal{N}(0, \sigma^2)$ under the MLE framework.

$$L(\sigma) = [\Phi(t_1; 0, \sigma^2)]^{c_1} \cdot [1 - \Phi(t_2; 0, \sigma^2)]^{c_2} \cdot \prod_{j=1}^{i} \varphi(x_j; 0, \sigma^2)$$

where $\Phi(x; \mu, \sigma^2)$ and $\varphi(x; \mu, \sigma^2)$ are the CDF and PDF of normal distribution with mean μ and variance σ^2.

Since we are only interested in the special case where two censoring points are symmetric($-t_1 = t_2 = t$), $L(\sigma)$ can be rewritten as follows

$$L(\sigma) = [\Phi(t; 0, \sigma^2)]^{m-i} \cdot \prod_{j=1}^{i} \varphi(x_j; 0, \sigma^2) \tag{2.8}$$

By taking the partial derivative of $L(\sigma)$, the estimate of σ, denoted by $\tilde{\sigma}(\|x_j\|)$, could be obtained by solving the following equation, where $\|x_j\| = \sqrt{\sum_{j=1}^{i} x_j^2}$ and $\xi = -t/\sigma$.

$$\frac{\partial(\ln L(\sigma))}{\partial\sigma} = -\frac{(m-i)\varphi(\xi;0,\sigma^2)}{\sigma\Phi(\xi;0,\sigma^2)} - \frac{i}{\sigma} + \frac{1}{\sigma^3}\sum_{j=1}^{i} x_j^2 = 0 \tag{2.9}$$

Recall that $\delta(o) = |h(q) - h(o)|$ follows $\mathcal{N}(0, d^2(o,q))$ according to Lemma 1. By regarding the search window $[-t,t]$ as the interval defined by two censoring points, the estimate of $d(o,q)$, denoted by $\tilde{\sigma}(\Delta^t(o))$, can be obtained using Equation (2.9) if we substitute $\|x_j\|$ with $\Delta^t(o)$. Similarly, by substituting $\|x_j\|$ with l_i, we can obtain the estimate of the radius in the d-dimensional space, denoted by $\tilde{\sigma}(l_i)$, for each physical hypersphere. Note that the variance of the estimate is a constant for given m, t and l_i as shown in [15].

It is easy to see that $\tilde{\sigma}(\|x_j\|)$ is a function of m, i, t and $\|x_j\|$. In order to express their relation more clearly, we transform Equation (2.9) into the following equation, where $G(i,\xi) = [\xi \cdot \frac{m-i}{i}) \cdot \varphi(\xi;0,\sigma^2) + \Phi(\xi;0,\sigma^2)]/[\xi^2 \cdot \Phi(\xi;0,\sigma^2)]$.

$$\|x_j\|^2 = it^2 G(i,\xi) \tag{2.10}$$

Next, we present an important property of $G(i,\xi)$.

Lemma 2. *For fixed $i > 0$, Equation $G(i,\xi) = 0$ has only one root $\xi_0 < 0$. In addition, when $\xi > \xi_0$, $G(i,\xi)$ is monotonically increasing with ξ and $\lim_{\xi\to 0^-} G(i,\xi) = +\infty$.*

By taking the derivative of $G(i,\xi)$, one can prove Lemma 2 readily, which implies that 1) a unique solution exists for Equation (2.10), and 2) $\tilde{\sigma}(\|x_j\|)$ increases monotonically with $\|x_j\|$ for given m, i and t.

(3)**Soundness of Virtual Hypersphere Partitioning.** In this subsection, we show the soundness of virtual hypersphere partitioning.

We can derive m estimated radii $\tilde{\sigma}(l_i)$ $(1 \le i \le m)$ for m physical hyperspheres. To make them equivalent to a unique virtual hypersphere, it is reasonable to set $\tilde{\sigma}(l_i)$ identical to each other as follows.

$$\tilde{\sigma}(l_i) = \tilde{\sigma}(l_j), 1 \le i, j \le m \tag{2.11}$$

Next we will show that, given P_*, m, t and s, Equation (2.11) along with Equation (2.7) gives a unique solution for each l_i. To start with, we need to establish the connection between $\tilde{\sigma}(\|x_j\|)$ with different i.

Lemma 3. *Given i full samples x_j ($1 \leq j \leq i$) out of m samples with respect to the censoring points $-t$ and t, let $\tilde{\sigma}(\sqrt{\sum_{j=1}^{i} x_j^2})$ ($i \in [1, m-1]$) be the estimate calculated using Equation (2.9), the inequality $\tilde{\sigma}(\sqrt{\sum_{j=1}^{i+w} x_j^2 + w \cdot t^2}) < \tilde{\sigma}(\sqrt{\sum_{j=1}^{i} x_j^2})$ holds for any $w \leq m - i$.*

Proof. We only need to consider the case $w = 1$ since more general cases can be proved by induction. Let $\xi_i = \frac{-t}{\tilde{\sigma}(\sqrt{\sum_{j=1}^{i} x_j^2})}$. By definition we know that $\sum_{j=1}^{i} x_j^2 = it^2 G(i, \xi_i)$ and $\sum_{j=1}^{i+1} x_j^2 = (i+1)t^2 G(i+1, \xi_{i+1})$. Since $|x_j| \leq t$, we have

$$(i+1)t^2 G(i+1, \xi_{i+1}) - it^2 G(i, \xi_i) \leq t^2$$

We can prove the following inequality (the details are tedious and omitted)

$$(i+1)t^2 G(i+1, \xi_i) - it^2 G(i, \xi_i) > t^2$$

As a result, $G(i+1, \xi_i) > G(i+1, \xi_{i+1})$ holds. Please note that $\xi_i > \xi_0$, $\xi_{i+1} > \xi_0$ and $G(i, \xi)$ increases monotonically with $\xi > \xi_0$ for fixed i. Thus, we have $\xi_i > \xi_{i+1}$, which leads to $\tilde{\sigma}(\sqrt{\sum_{j=1}^{i} x_j^2}) > \tilde{\sigma}(\sqrt{\sum_{j=1}^{i+1} x_j^2 + t^2})$ immediately. □

With the help of Lemma 3, we can readily prove the following proposition by contradiction.

Proposition 1. *If $(l_1, l_2, ..., l_m)$ satisfies Equation (2.11), we have $l_i < l_j$ for any $1 \leq i < j \leq m$.*

Now we are in the position to show the uniqueness of $(l_1, l_2, ..., l_m)$ under the constraints of Equation (2.11) and Equation (2.7).

Theorem 1. *Given P_*, m, t and s, there exists a unique sequence $(l_1, l_2, ..., l_m)$ such that Equation (2.11) and Equation (2.7) hold.*

(Sketch). This theorem can be proved by contradiction according to Proposition 1, the facts that $\sum_{i=0}^{m} C(m,i)\ p^i(1-p)^{m-i} = 1$ for $0 < p < 1$, and functions $\mathcal{G}_i^t(l_i)$ and $\tilde{\sigma}(l_i)$ increase monotonically with l_i. □

By Theorem 1 and the fact $\tilde{\sigma}(\|x_j\|)$ increases monotonically with $\|x_j\|$, we can prove that the condition $\Delta^t(o) \leq l_i$ for any $1 \leq i \leq m$ (Equation (2.5)) is equivalent to $\tilde{\sigma}(\Delta^t(o)) \leq \tilde{\sigma}(l_i)$ readily, which justifies the soundness of virtual hypersphere partitioning.

2.4.2.5 Calculate the Base Hypersphere Radii

Based on Theorem 1, we could come up with a simple algorithm to find NN if s_* were known somehow beforehand. Specifically, one can compute $(l_1, l_2, ..., l_m)$ first. Then by examining all points such that $\Delta^t(o) \leq l_{r_t(o)}$, it is guaranteed that one can achieve P_* in finding o_*.

Unfortunately, s_* is unavailable in the first place. As a workaround, we first derive the *base hypersphere radii* $l_i^{t_0}$ such that $p_{\mathcal{L}}^{t_0}(1) = P_*$ using Algorithm 2, under the assumptions that the *base search window* is equal to $[-t_0, t_0]$ and $d(o_*, q) = 1$. Thanks to Proposition 2, we are able to scale the search window and hypersphere radii properly and achieve the desirable success probability for any s.

Proposition 2. *$p_{\mathcal{L}_s}^{st_0}(s) = p_{\mathcal{L}_1}^{t_0}(1)$ $(s > 0)$, where $\mathcal{L}_1 = \{l_1^{t_0}, l_2^{t_0}, \ldots, l_m^{t_0}\}$ and $\mathcal{L}_s = \{sl_1^{t_0}, sl_2^{t_0}, \ldots, sl_m^{t_0}\}$*

Proof. This proposition can be proved easily by the facts that, for any $s > 0$, $\mathcal{G}_i^{st_0}(sl_i^{t_0}; 0, s^2) = \mathcal{G}_i^{t_0}(l_i^{t_0}; 0, 1)$ and $p(s)$ under $t = st_0$ is equal to $p(1)$ in the case of $t = t_0$. □

Proposition 2 suggests that, for any point o such that $d(q, o) = s$, we can simply extend the search window and hypersphere radii from the base ones to $[-st_0, st_0]$ and $(sl_1^{t_0}, sl_2^{t_0}, \ldots, sl_m^{t_0})$ respectively to achieve P_* for o. In addition to ensuring the success probability, the other benefit of Proposition 2 is that we do not have to evaluate $(l_1, l_2, \ldots, l_m)$ at runtime as the search windows grow.

Based on Proposition 2, VHP can work as follows. Starting with the base hypersphere radii $\{l_1^{t_0}, l_2^{t_0}, \ldots, l_m^{t_0}\}$, VHP enlarges the hypersphere radii in a coordinated way by multiplying $l_i^{t_0}$ with $\frac{t}{t_0}$, where $2t$ is the new window size. By examining all points such that $\Delta^t(o) \leq l_{r_t(o)}$, VHP maintains o_{min}, which is the nearest point to q found so

Algorithm 2: Compute the base radii ( )

Input: m is the number of hash functions; $2t_0$ is the search window size, $s_* = 1$ is the distance between q and its NN and P_* is the expected success probability;

Output: radii $(l_1^{t_0}, l_2^{t_0}, ..., l_m^{t_0})$

1 Solve Equation (2.11) and Equation (2.7) to get a unique solution $(l_1^{t_0}, l_2^{t_0}, ..., l_m^{t_0})$;

2 return $(l_1^{t_0}, l_2^{t_0}, ..., l_m^{t_0})$

far. As will be proved in Section 2.4.3.1, the probability that VHP finds o_{min}, which is equal to $p_{\mathcal{L}}^{d(q,o_{min})t_0}(d(q, o_{min}))$, is a lower bound of the probability of getting o_*. Then, by setting $p_{\mathcal{L}}^{d(q,o_{min})t_0}(d(q, o_{min})) \geq P_*$ as the termination condition, we can find o_* with probability P_* for sure, which means that VHP will succeed with probability at least P_*. In practice, we use the following Inequality (2.12) as the termination condition, which is much cheaper to evaluate.

$$d(q, o_{min}) \geq \frac{t}{t_0} \tag{2.12}$$

The equivalency between Inequality (2.12) and $p_{\mathcal{L}}^{d(q,o_{min})t_0}(d(q, o_{min})) \geq P_*$ can be proved readily using Proposition 2.

The following proposition, which can be proved using Equation (2.10), shows that the radius of the virtual hypersphere is still unique after scaling.

Proposition 3. *For any* $1 \leq i, j \leq m$ *and* $s > 0$, *if* $\tilde{\sigma}(l_i^{t_0}) = \tilde{\sigma}(l_j^{t_0})$ *then* $\tilde{\sigma}(l_i^{st_0}) = \tilde{\sigma}(l_j^{st_0})$ *and* $\tilde{\sigma}(l_i^{st_0}) = s\tilde{\sigma}(l_i^{t_0})$.

2.4.3 Theoretical Analysis

2.4.3.1 Probability Guarantee for NN Search

In this subsection we show that, by extending the search windows and increasing the radii of the hyperspheres coordinately, VHP is guaranteed to find the NN of q with probability at least P_*. To prove this, we need to introduce an *Oracle* algorithm VHP$_o$ first. Simply put, VHP$_o$ is the same as VHP except that it is told by the Oracle the distance between o_* and q. Obviously, VHP$_o$ finds o_* with probability P_* for sure as discussed in the last subsections.

According to the terminating condition in Algorithm 1 (Line 2), $t_* = s_* t_0$ and $l_i^{t_*} = s_* l_i^{t_0}$ when VHP$_o$ terminates, where $s_* = d(o_*, q)$. Similarly, $t_a = s_a t_0$ and

$l_i^{t_a} = s_a l_i^{t_0}$ when the actual VHP terminates, where $s_a = d(o_{min}, q)$. Since $s_a \geq s_*$, we have $t_a \geq t_*$ and $l_i^{t_a} \geq l_i^{t_*}$. In other words, the final search window size and radii imposed by VHP are greater than those by VHP$_o$. To show the probability with which VHP finds o_* is greater than that of VHP$_o$, we need to prove the following Lemma first.

Lemma 4. *Given m random B^+-trees, dataset $\mathcal{D}$ and two search windows of sizes $2t_1$ and $2t_2$ ($t_2 > t_1$), $\forall o \in \mathcal{D}$, it holds that $\Delta^{t_2}(o) \leq \frac{t_2}{t_0} l^{t_0}_{r_{t_2}(o)}$ if $\Delta^{t_1}(o) \leq \frac{t_1}{t_0} l^{t_0}_{r_{t_1}(o)}$.*

Proof. Assume the radii associated with $l^{t_1}_{r_{t_1}(o)}$ and $l^{t_2}_{r_{t_2}(o)}$ are $\tilde{\sigma}(l^{t_1}_{r_{t_1}(o)})$ and $\tilde{\sigma}(l^{t_2}_{r_{t_2}(o)})$, respectively. To prove this Lemma, we only need to show that $\tilde{\sigma}(\Delta^{t_2}(o)) \leq \tilde{\sigma}(l^{t_2}_{r_{t_2}(o)})$ if $\tilde{\sigma}(\Delta^{t_1}(o)) \leq \tilde{\sigma}(l^{t_1}_{r_{t_1}(o)})$.

By definition $\Delta^{t_1}(o)$ and $\Delta^{t_2}(o)$ are equal to $\sqrt{\sum_{h_i \in \mathcal{I}_{t_1}(o)} \delta_i^2(o)}$ and $\sqrt{\sum_{h_i \in \mathcal{I}_{t_2}(o)} \delta_i^2(o)}$, respectively. Please note that $\mathcal{I}_{t_1}(o)$ is a subset of $\mathcal{I}_{t_2}(o)$. Let $w = |\mathcal{I}_{t_2}(o)| - |\mathcal{I}_{t_1}(o)|$. Suppose there are two points $o_\dagger$ and $o_\ddagger$ such that (1) $\mathcal{I}_{t_2}(o_\dagger) = \mathcal{I}_{t_1}(o)$ and $\delta_i(o_\dagger) = \frac{t_2}{t_1}\delta_i(o)$ for $i \in \mathcal{I}_{t_1}(o)$; (2) $\mathcal{I}_{t_2}(o_\ddagger) = \mathcal{I}_{t_2}(o)$ and $\delta_i(o_\ddagger) = \frac{t_2}{t_1}\delta_i(o)$ for $i \in \mathcal{I}_{t_1}(o)$ and, (3) $\delta_i(o_\ddagger) = \delta_i(o)$ for $i \in \mathcal{I}_{t_2}(o) - \mathcal{I}_{t_1}(o)$. By definition we have $\Delta^{t_2}(o_\dagger) = \sqrt{\sum_{h_i \in \mathcal{I}_{t_1}(o)} (\frac{t_2}{t_1}\delta_i(o))^2}$. It is easy to see that $\tilde{\sigma}(\Delta^{t_2}(o_\dagger)) \leq \frac{t_2}{t_1}\tilde{\sigma}(l^{t_1}_{r_{t_1}(o)})$ by Equation (2.10). Then, with the help of Lemma 3 we know that $\tilde{\sigma}(\Delta^{t_2}(o_\ddagger)) \leq \tilde{\sigma}(\Delta^{t_2}(o_\dagger))$ since $(\Delta^{t_2}(o_\dagger))^2 + wt_2^2 \geq (\Delta^{t_2}(o_\ddagger))^2$. Recall that $\tilde{\sigma}(\Delta^t(o))$ is an increasing function of $\Delta^t(o)$ for fixed t and i, and thus we have $\tilde{\sigma}(\Delta^{t_2}(o)) \leq \tilde{\sigma}(\Delta({}^{t_2}o_\ddagger))$ considering $\Delta^{t_2}(o) \leq \Delta^{t_2}(o_\ddagger)$. By putting these inequalities together, it holds that $\tilde{\sigma}(\Delta^{t_2}(o)) \leq \frac{t_2}{t_1}\tilde{\sigma}(l^{t_1}_{r^{t_1}(o)})$. According to Proposition 3, we have $\frac{t_2}{t_1}\tilde{\sigma}(l^{t_1}_{r^{t_1}(o)}) = \tilde{\sigma}(l^{t_2}_{r^{t_2}(o)})$, thus complete this proof. □

Lemma 4 indicates that, as VHP increases the radius of the virtual hypersphere dynamically, the candidate set under $[-t_1, t_1]$ is always a subset of that under $[-t_2, t_2]$ for any $t_2 \geq t_1$. Thus, we proved the self-consistency of the virtual hypersphere partitioning.

Now we are ready to show the correctness of VHP based on Lemma 4.

Theorem 2. *Algorithm 1 returns the NN of q with probability at least P_*.*

Proof. We first define the following two events:

E_1: o_* is found by VHP$_o$.

E_2: o_* is found by VHP.

As discussed earlier, $t_a \geq t_*$ and $l_i^{t_a} \geq l_i^{t_*}$ hold. By Lemma 4 we know that the points visited by VHP$_o$ are contained in a subset of those examined by VHP, and thus $P[E_2] \geq P[E_1]$ follows. By the fact that VHP$_o$ is guaranteed to find o_* with P_*, we have $P[E_2] \geq P_*$, and thus complete the proof. □

2.4.3.2 Extension for c-k-ANN Search

To support the c-k-ANN search, it is sufficient to replace the terminating condition (line 2 in Algorithm 1) with $\frac{d(o_{min_k},q)}{c} > \frac{t}{t_0}$, where o_{min_k} is the k-th nearest neighbor of q found so far. VHP outputs k neighbors, i.e. $o_{min_1}, o_{min_2}, ..., o_{min_k}$ instead of o_{min}. In this way, VHP supports probabilistic NN and c-k-ANN search in the same framework.

Next, we will show that VHP returns c-ANN ($c > 1$) of q with probability at least P_*. For clarity of presentation, we refer to VHP$_c$ ($c > 1$) as the c-ANN version of Algorithm 1. For the c-k-ANN version of VHP, the probability guarantee can be proved in the similar vein and omitted due to space limitation.

Theorem 3. *VHP$_c$ returns a c-ANN of q with probability at least P_**

Proof. Besides the two events E_1 and E_2 defined in Theorem 1, we need the following three events:

E_3: VHP$_c$ terminates before the search window extends to $[-t_*, t_*]$, where $t_* = s_* t_0$.

E_4: a c-ANN of q is found.

E_5: o_* is found.

Let $[-t_c, t_c]$ denote the final search window when VHP$_c$ terminates. Obviously we have $P[E_4] = P[E_4|E_3]P[E_3] + P[E_4|\bar{E_3}]P[\bar{E_3}]$. To prove the theorem, we only need to show $P[E_4|E_3] \geq P[E_1|E_3]$ and $P[E_4|\bar{E_3}] \geq P[E_1|\bar{E_3}]$ since $P[E_1] = P_*$. The former inequality holds by the fact $P[E_4|E_3]$ equals 1, which can be proved by contradiction. Particularly, assume VHP$_c$ terminates before the search window reaches $[-t_*, t_*]$, i.e. $t_c < t_*$, and did not find any c-ANN. Then we must have $d(o_{min}, q) > c \cdot d(o_*, q)$. According to the terminating condition of Algorithm 1, VHP$_c$ terminates only if $\frac{d(o_{min},q)}{c} \leq \frac{t_c}{t_0}$, which implies that $t_c > t_*$, and thus a contradiction. The latter inequality holds because $P[E_4|\bar{E_3}] \geq P[E_5|\bar{E_3}]$ (with the same search window, a c-ANN must be identified if o_* is found) and $P[E_5|\bar{E_3}] \geq P[E_1|\bar{E_3}]$ (Lemma 4). Thus we conclude. □

2.4.4 Discussion

2.4.4.1 Complexity analysis

In the worst case, the time complexity of VHP is $O(n(m+d))$. As will be discussed in Section 2.4.5, m is far smaller than n and thus can be regarded as a small constant. Thus, the worst case time complexity is reduced to $O(nd)$, which is consistent with the common wisdom on the hardness of NN search in high-dimensional spaces. However, as shown in Section 2.4.5, the actual performance of VHP is far better than the worst case one. The space consumption consists of two parts: the space for storing the data set $O(nd)$ and the space of index $O(mn)$. Thus, the total space complexity of VHP is $O(n(m+d))$.

VHP can easily support updating (insertion, deletion and modification) due to the utilization of B^+-trees. It is notable that, although $\{l_i^{t_0}\}$ need to be determined beforehand, their values only depend on the user-specified parameters m, P_* and t_0. Hence, the updates, which might affect the data distribution, has no impact on $\{l_i^{t_0}\}$.

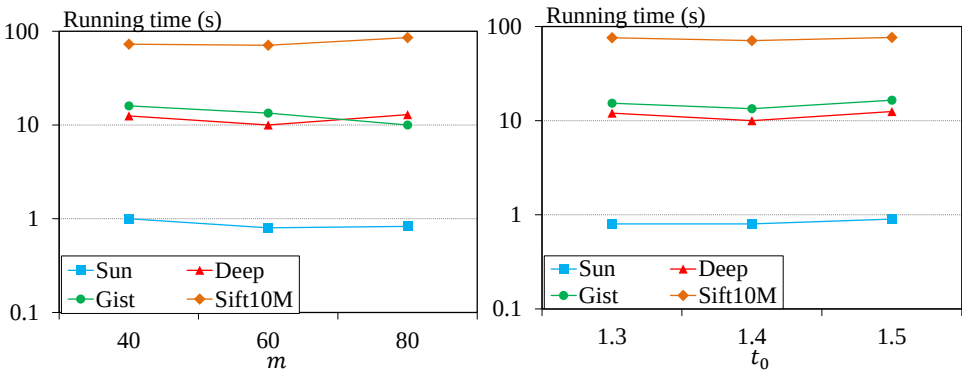
2.4.4.2 Comparison with existing methods

While both VHP and PAC-NN use hyperspheres, there are two fundamental differences between them: (1) PAC-NN uses a single physical hypershpere in the original space directly, whereas VHP employs multiples physical hypersperes in projection subspaces to emulate one virtual hypershpere in the original feature space. (2) PAC-NN and VHP deliver different kinds of theoretical guarantee. Specifically, PAC-NN supports data-dependent probability guarantee, which needs data distribution information around each query. In contrast, as an LSH-sytle algorithm, VHP has no assumption on data distribution, and thus is data independent.

2.4.5 Experimental results

2.4.5.1 Experiment Setup

- **SRS [51].** We use SRS-2, a variant of SRS, in our experiments because SRS-2 supports arbitrary $c \geq 1$ and $P_* < 1$ like VHP. m was set to the default value, i.e. 6, as suggested in [35, 51] and P_* was set to 0.9 by default.



(a) Different m **(b)** Different t_0

Figure 2.4: The impact of different parameters

- **QALSH [26].** The success probability of QALSH was set to $(1/2-1/e)$, as suggested in [26]. m was computed using the method described in [26] as well. c was set to 2 by default unless stated otherwise.

- **HD-index [5].** HD-index is a recently proposed representative of non-LSH methods (without quality guarantee) for disk-based ANN search. All internal parameters of HD-index were adjusted to be experimentally optimal, as suggested in [5].

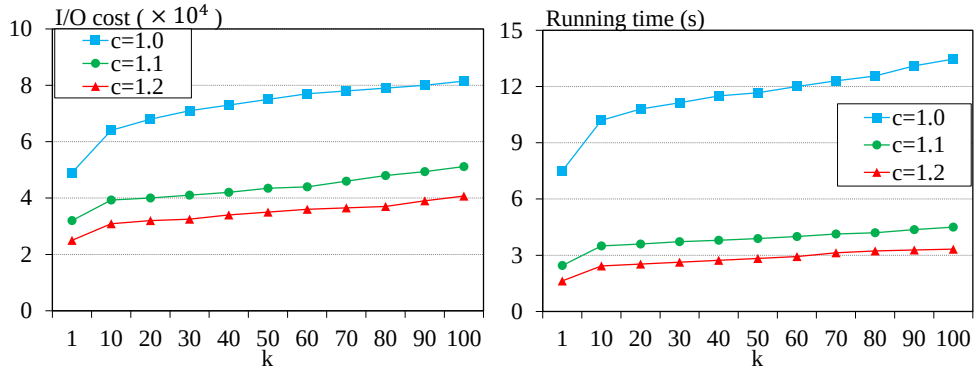
- **VHP.** In all experiments, P_* was set to the same value as that of SRS, i.e. $P_* = 0.9$. Optimal t_0 and m depend on the concrete data distributions. As will be shown in Section 2.4.5.2, VHP obtains near optimal performance when $t_0 = 1.4$ and $m = 60$ for all datasets we experimented with. Thus, t_0 and m were set to 1.4 and 60 by default, respectively.

For all methods, we used their external-memory versions. To be specific, for SRS we use the R-tree based external-memory version, which is originally proposed to support billion scale datasets on a commodity PC [51]. As for HD-index and VHP, we build RDB-tree/B^+-tree by bulk loading such that they can scale in our setting.

We used six publicly available real-world datasets as listed below. The page size was set to 4KB. The value of k is fixed to 100 unless stated otherwise.

- **Sun**[1] consists of 79106 512-dim GIST features of images.

[1] http://groups.csail.mit.edu/vision/SUN/



(a) Gist, I/O Cost **(b)** Gist, Running time

Figure 2.5: The performance of VHP under different approximation ratios

- **Deep**[1] contains 1 million 256-dim deep neural codes of natural images obtained from the activations of a convolutional neural network.
- **Gist**[2] is an image dataset which contains about 1 million 960-dim data points.
- **Sift10M**[3] consists of 10 million 128-dim SIFT vectors.
- **Sift1B**[4] consist of 1 billion 128-dim SIFT vectors.
- **Deep1B**[5] consist of 1 billion 96-dim DEEP vectors.

2.4.5.2 Parameter setting of VHP

Parameters t_0 and m have important impact on the performance and index size of VHP. We empirically determine the near optimal t_0 and m. Partial statistics over four datasets under different combinations of t_0 and m are shown in Figure 2.4, where $c = 1.0$ and $k = 100$. According to the results, we can see:

(1) $t_0 = 1.4$ is an appropriate choice under which VHP runs fastest for four datasets when m is fixed. In fact, the performance degrades dramatically for too small or too large t because the collision probability tends to 0 or 1 for all points.

(2) As for m, we can see that VHP works well when $m = 60$. It is notable that the performance of VHP on Gist can be better in the case of $m = 80$. This is because the dimensionality of Gist is much higher (dim 960) and more hash functions may help to distinguish nearest neighbors better.

[1]https://yadi.sk/d/I_yaFVqchJmoc
[2]http://corpus-texmex.irisa.fr/
[3]https://archive.ics.uci.edu/ml/datasets/SIFT10M
[4]http://corpus-texmex.irisa.fr/
[5]https://github.com/facebookresearch/faiss/tree/master/

Table 2.2: Comparison of index sizes. (CR means crash in the indexing phase)

	SRS	QALSH	HD-index	VHP
Sun	3.1M	21.2M	250M	18.9M
Deep	36.5M	350.9M	2.0G	228.5M
Gist	36.8M	350.9M	18.1G	228.5M
Sift10M	524.7M	4.1G	10.2G	2.5G
Sift1B	39.2G	CR	1.2T	251G
Deep1B	39.5G	CR	0.9T	251G

Based on these observations, we chose $m = 60$ and $t_0 = 1.4$ as the default for all datasets we experimented with.

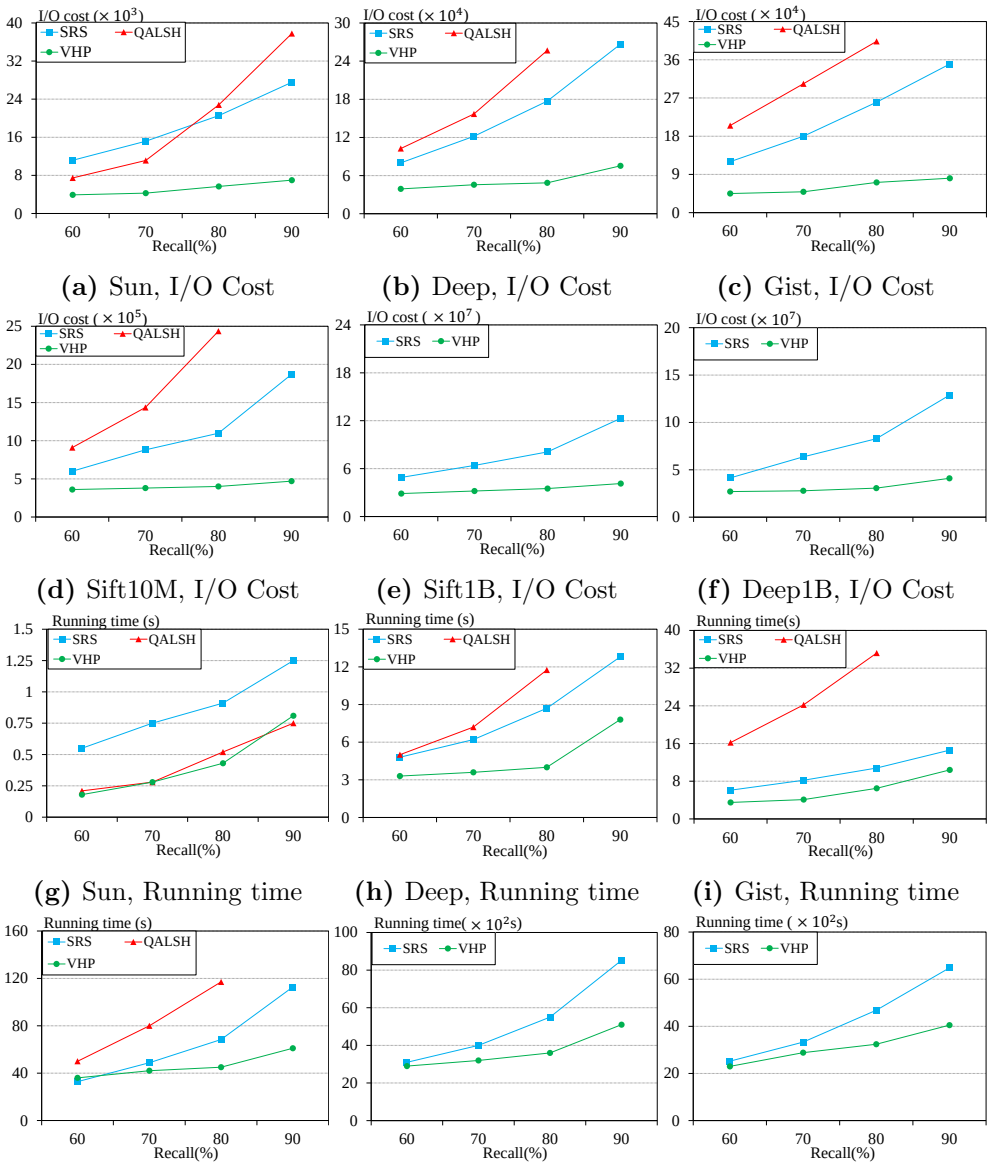
2.4.5.3 The Effect of Approximation Ratio

Like most LSH-based methods, VHP can trade the result quality for speed by tuning the approximation ratio c. Figure 2.5 depicts the performance of VHP on Gist under different approximation ratios (similar trends were observed on other datasets). As one can see, both I/O cost and running time of VHP increases with k. This is because the larger k is, the more points have to be visited to achieve the desirable answer quality. Also, by setting larger c, the searching process can be accelerated at the cost of accuracy.

The overall ratios (answer quality) of VHP for $k = 100$ under $c = 1.0$, $c = 1.1$ and $c = 1.2$ are around 1.001, 1.02 and 1,04, respectively. We can see the real overall ratio is much smaller than the corresponding approximation ratio. The reason is that the probability guarantee of VHP is obtained using the worst-case analysis, which is often not the case of real datasets.

2.4.5.4 Index Size, Indexing Time and Memory Consumption

Table 2.2 lists the index size of four methods over the six datasets. We can see that the index size of SRS is the smallest whereas HD-index requires the maximum space consumption. The index size of VHP is around $6-7$ times greater than that of SRS, but around 1.5 times smaller than that of QALSH. This is because, for LSH-based methods, the index size is proportional to the number of hash functions. QALSH crashed on large datasets due to the out of memory exception.



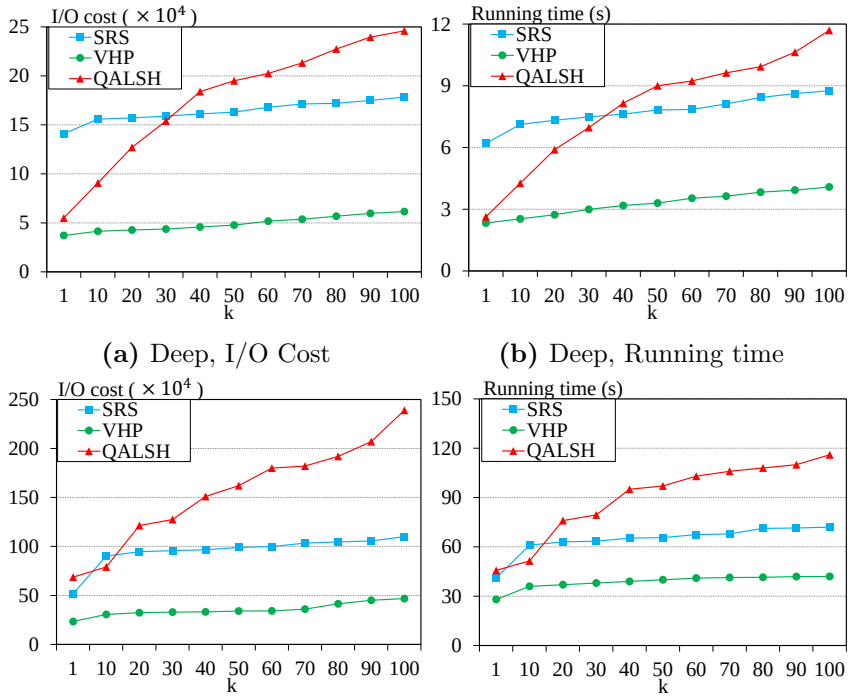
(a) Sun, I/O Cost **(b)** Deep, I/O Cost **(c)** Gist, I/O Cost

(d) Sift10M, I/O Cost **(e)** Sift1B, I/O Cost **(f)** Deep1B, I/O Cost

(g) Sun, Running time **(h)** Deep, Running time **(i)** Gist, Running time

(j) Sift10M, Running time **(k)** Sift1B, Running time **(l)** Deep1B, Running time

Figure 2.6: The comparison on the accuracy-efficiency tradeoffs of VHP, SRS and QALSH



(a) Deep, I/O Cost **(b)** Deep, Running time

(c) Sift10M, I/O Cost **(d)** Sift10M, Running time

Figure 2.7: The performances of VHP under different k at recall 80%

Among all methods, the indexing time of VHP is the smallest, followed by QALSH, SRS and HD-index. Take the largest dataset Sift1B as an example, VHP takes 18 hours for indexing while SRS and HD-index need 4 days and 11 days, respectively. This is because B^+-tree costs less time than R-tree and RDB-tree in index construction.

As for the main memory consumption in the indexing phase, we report the results on Sift1B as follows: SRS consumes 1.1GB, VHP takes 1.9GB and HD-index needs 97MB. Thus, the indexing phase of all three methods can be accomplished successfully on a commodity PC.

2.4.5.5 VHP vs. LSH-based Methods

In this section, we compare VHP with the baseline LSH-based methods SRS and QALSH. In order to make the comparison more reasonable, we fix the expected recall and measure how much running time and I/O cost it takes for three LSH-based methods in this paper. Such a comparison is feasible because all of them can make the tradeoff between cost and answer quality by tuning the approximation ratio c.

2.4.5.6 Experimental results under the same recall

In Figure 2.6, the target recalls are set to 60%, 70%, 80%, 90% because they are high enough for the practical use. According to the results, we have following observations.

(1) At the same precision level, VHP needs only around $\frac{1}{7}$ to $\frac{1}{4}$ I/O cost of QALSH and $\frac{1}{3}$ to $\frac{1}{2}$ I/O cost of SRS. The reason is that VHP uses a relatively small index and more selective filtering method. Accordingly, VHP achieves up to 2x speedup over SRS and up to 4x speedup over QALSH. Please note that the speedup is not exactly proportional to the gain over I/O cost due to the (uncontrolled) impact of caching at different levels.

(2) The superiority of VHP over the other two methods becomes relatively less significant at low recall, say 60%. This is because, as the target recall is getting lower, it becomes easier for all three methods to find answers satisfying the less strict requirement, which in turn reduces the differences among their performance. In practice, however, end users often expect high answer quality, where VHP can perform very well as shown above. On datasets Sun and Gist of high dimensionality, VHP performs better in speed than it does on those of low dimensionality, which indicates that VHP is more preferable for high dimensional datasets.

2.4.5.7 Experimental results under different k

The results above were all obtained under $k = 100$, we also compared three methods for different k in Figure 2.7. Due to space limitation, we only list the results of Sift10M and Deep under target recall 80%. Similar trends were observed on other datasets. From the results, we can see that (1) for all k, VHP beats SRS and QALSH in both running time and I/O cost, which suggests that the performance of VHP is very stable as k varies. (2) As k increases, the cost of QALSH increases dramatically while the performance of VHP and SRS degrades rather smoothly. This indicates that VHP and SRS are more promising than QALSH for the k-ANN search where k is large.

2.5 R2LSH: LSH in two dimensional subspaces

2.5.1 Motivation

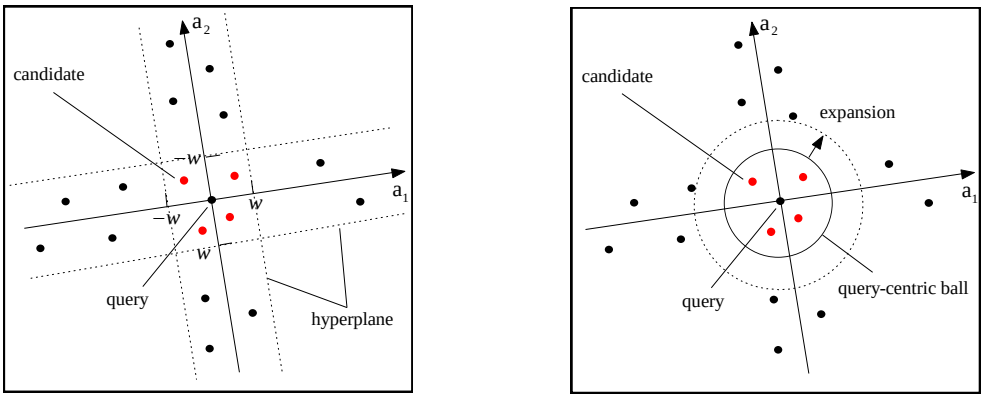
First, let us analyse the limitations of QALSH from another perspective. From the working mechanism of QALSH introduced earlier, it can remove most false objects by

collision testing. Thus, the size of the candidate set is fairly small, which significantly reduces the I/O cost of retrieving candidates. However, before identifying candidates, QALSH consumes a significant amount of I/Os to scan those objects colliding with q owing to the inefficiency of one-dimensional hash buckets. Consider Figure 2.8(a) as an example. For simplicity, we suppose there are only two hash functions h_{a_1} and h_{a_2}. The threshold τ is set to two, which implies that only those objects that collide with q in two hash buckets simultaneously can be considered as candidates (red points). To obtain these candidates, QALSH has to scan all the objects (black points) that lie between two hyperplanes at a distance of $2w$ from each other. Because the number of scanned objects can be exceptionally large and these objects may appear in multiple hash buckets, the scanning process incurs substantial I/O cost. Almost all the I/O cost incurred in the query phase of QALSH originates from this scanning.

Thus, to reduce the I/O cost of scanning, we propose a more efficient indexing and searching scheme named R2LSH. The fundamental concept underlying R2LSH is illustrated in Figure 2.8(b). Similarly, we again consider the case of the two hash functions h_{a_1} and h_{a_2}. First, using the mapping $o \mapsto (h_{a_1}(o), h_{a_2}(o))$ (where o is an arbitrary object in the dataset), we can construct a two-dimensional space M that contains images of all the objects, such that the coordinate of the mapped o in M is $(h_{a_1}(o), h_{a_2}(o))$. Then, rather than use two hash buckets, we construct a query-centric ball of radius r in this space and ensure that all the objects within this ball can be scanned by introducing a suitable index structure (a small number of objects outside the ball may also be scanned). All the scanned objects are considered as candidates (red points). Evidently, query-centric balls work more selectively than query-centric hash buckets because only those objects whose projections are close to the query's projections on both the axes are likely to be scanned. Therefore, we can circumvent substantial I/O cost in scanning false objects, which also leads to a smaller m (around 40) of R2LSH than that of QALSH (around or over 100) in practice.

2.5.2 Overview

An overview of R2LSH is depicted in Figure 2.9 and related notations are listed in Table 2.3. In the indexing phase, we index all the data objects by multiple B^+-trees as follows. First, we construct m two-dimensional projected spaces $\{M_i\}_{i=1}^{m}$ and map objects



(a) Hash bucket **(b)** Ball (dim 2)

Figure 2.8: Query-centric Hash bucket ($m = 2$, $\tau = 2$) vs. Query-centric Ball ($\tau = 1$).

into these projected spaces. Then, we construct a suitable index structure to characterize data distributions in $\{M_i\}$. Because the projected space is not one-dimensional, we cannot directly use an individual B$^+$-tree to record the complete information of objects in it. To overcome this challenge, in each M_i, we select α reference objects $\{C_j^i\}_{j=1}^{\alpha}$ and construct β rings $\{R_{j,l}^i\}(1 \leq l \leq \beta)$ sharing each center C_j^i so as to contain all the objects in M_i. In addition, we select an additional reference vector v_i in each M_i. For each ring $R_{j,l}^i$, we translate v_i such that its starting point coincides with C_j^i and compute the reference angle of each object in $R_{j,l}^i$ to v_i. Then, we can construct a B$^+$-tree $T_{j,l}^i$ corresponding to $R_{j,l}^i$ such that the reference angle of each object in $R_{j,l}^i$ is considered as the key of this object in $T_{j,l}^i$. All such B+-trees form the index structure of R2LSH.

In the query phase, given query q, we set up a query-centric ball $B(g_i(q), r)$ in each M_i. Here, the center $g_i(q)$ is the image of q in M_i, and the radius r is initially set to zero. As the search proceeds, we increase r (for each ball) in steps of Δr and scan objects newly included in the expanded balls. Two conditions play important roles in this process. The first is the candidate condition, which determines whether the scanned object can be presently considered as a candidate. Similar to C2LSH [20] and QALSH [26], the candidate condition is obtained by collision testing. The second is the termination condition. We stop the expansion of balls once and return results from candidates if this condition is satisfied. This condition is intimately related to our probability guarantee, which will be introduced in detail in Sec. 2.5.4.

Table 2.3: Notations of R2LSH

Notation	Explanation
d	the dimensionality of data object
c	the approximation ratio specified by users
δ	the error rate specified by users
o^*, o_i^*	the first and i th nearest neighbor of q
$\hat{o}, \hat{o}_i$	the first and i th nearest candidate of q
$s^*, \hat{s}$	$s^* = \|o^* - q\|$ and $\hat{s} = \|\hat{o} - q\|$
m	the number of generated projected spaces
M_i	the i th projected space ($1 \leq i \leq m$)
α	the number of reference objects in a single projected space
β	the number of rings centered at each reference object
$g_i(o)$	the image of o in M_i
$\Delta_i(o)$	$\Delta_i(o) = \|g_i(o) - g_i(q)\|$
$B(g_i(q), r)$	the query-centric ball centered at $g_i(q)$, of radius r
$\#Col(o; r)$	the collision number of o when the radii of query-centric balls are commonly set to r
C_j^i	the i th reference object in M_i
$R_{j,l}^i$	the l th ring centered at C_j^i in M_i
v_i	the reference vector in M_i
$T_{j,l}^i$	the B+-tree contains the information of objects in $R_{j,l}^i$
τ	the threshold on collision numbers
λ	$\lambda = \sqrt{\frac{1}{2m} \log \frac{1}{\delta}} + \frac{\tau}{m}$
Φ	CDF of the chi-square distribution $\chi^2(2)$

In order to increase the understandability, we present an example in Figure 2.10). In this example, three sets of co-centered rings are constructed such that each set includes part of the projected objects in ring. A ring set $\{R_{j,l}\}$ centered at C_j constructs a B+-tree set $\{T_{j,l}\}$. A B+-tree $T_{j,l}$ is constructed over the projected objects in $R_{j,l}$ with respect to their angles to a reference vector v. For example, $o_1 \in R_{1,2}$ is included only in $T_{1,2}$, and the location is $\pi/3$ when a query is mapped into this space as q. A query-centric circle of specified radius r (a red circle) is simulated by those B+-trees as follows. The angle of q to v is examined, and each $T_{j,l}$ is searched. Among these, only the objects in the search interval of q need to be examined (for counting). Presently, o_4 is identified. More objects can be identified by expanding the query-centric ball (the dotted circle) and increasing the size of the search intervals (dotted rectangles).

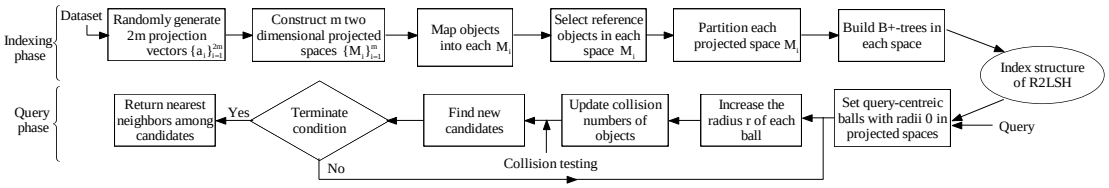


Figure 2.9: Overview of R2LSH

2.5.3 Indexing phase

In this section, we introduce a method to construct the index structure of R2LSH (Algorithm 12). The whole indexing phase can be divided into four steps: 1) construction of projected spaces, 2) partition of projected spaces, 3) reference vector selection, and 4) indexing objects by B^+-trees. These are introduced below in that order.

2.5.3.1 Construction of projected spaces

First, we generate $2m$ hash functions $\{h_{a_i}\}_{i=1}^{2m}$, where the vectors $\{a_i\}_{i=1}^{2m}$ are i.i.d samples from the Gaussian distribution $\mathcal{N}(0, I_d)$. For any object $o \in \mathbb{R}^d$, we can compute its hash values as $(h_{a_1}(o), h_{a_2}(o), \cdots, h_{a_{2m}}(o))$. Furthermore, we couple successive pairs of values as $g_i(o) = (h_{a_{2i-1}}(o), h_{a_{2i}}(o))$ to construct a two-dimensional projected space $M_i \in \mathbb{R}^2$ $(1 \leq i \leq m)$. o is mapped to this space by g_i as $g_i(o) \in M_i$.

2.5.3.2 Partition of projected spaces

Let us focus on a projected space M_i. The partition process is divided into two steps: 1) select the necessary number of reference objects in M_i, and 2) construct concentric rings centered at each reference object such that each object in M_i is uniquely contained in one of the rings. We consider the following two strategies for selecting reference objects.

Cluster-based selection. If the dataset $\mathcal{D}$ is well clustered, it is highly likely that the cluster structure is maintained uniform in M_i. Therefore, M_i can be considered as a subspace of the original space. In this case, we determine α clusters in M_i by clustering. Then, the centroid of each cluster is considered as a reference object. By clustering, we can make each object lie in a large ring centered at a certain cluster centroid. Finally, for each centroid, we equi-partition its corresponding ring into β concentric sub-rings $\{R_{j,l}^i\}_{l=1}^{\beta}$ centered at C_j^i such that each $R_{j,l}^i$ $(1 \leq l \leq \beta)$ contains an identical number of objects.

Simple selection. In certain cases, we need to address dynamic datasets with distributions drifting over time. In this case, for convenient update, we select the center of the data space as a reference object ($\alpha = 1$). Then, we partition its corresponding ring into β sub-rings of equal area.

2.5.3.3 Reference Vector Selection

In this section, we describe a method to determine a reference vector v_i in each M_i ($1 \leq i \leq m$). Because objects in a ring are distinguished only by their angles to the reference vectors in B$^+$-trees, the angle between any two v_i's is expected to be close to $\pi/2$ to the extent feasible. First, note that two axes of M_i are constructed by randomly generated vectors a_{2i-1} and a_{2i}. These can be represented as $\sum_{j=1}^{n} a_j^i \boldsymbol{e_j}$ and $\sum_{j=1}^{n} b_j^i \boldsymbol{e_j}$, respectively. Here, $\{\boldsymbol{e_j}\}_{j=1}^{d}$ is the natural basis of the original d-dimensional space, and $\{a_j^i\}$, $\{b_j^i\}$ are coefficients. Thus, v_i can be represented as $v_i(\lambda_1^i, \lambda_2^i) = \sum_{j=1}^{m} (\lambda_1^i a_j^i + \lambda_2^i b_j^i)\boldsymbol{e_j}$, where λ_1^i and λ_2^i are coefficients to be determined.

Then, in the average sense, we adopt the following strategy to determine $\{v_i\}$ sequentially in the greedy way.

Step 1: For the first projected space M_1, we arbitrarily select a normalized vector in M_1 as v_1.

Step 2: Suppose that we have determined $i-1$ reference vectors $v_1, \ldots, v_{i-1}$. For v_i, we require $\sum_{j=1}^{i-1} \langle v_i(\lambda_1^i, \lambda_2^i), v_j \rangle = 0$. In addition, we require $(\lambda_1^i)^2 + (\lambda_2^i)^2 = 1$ and $\lambda_1^i \geq 0$ for the norm of v_i to have no effect on the angle computation. It is evident that under the above constraints, λ_1^i and λ_2^i can be obtained uniquely, and thereby, v_i can be determined. Then, we repeat Step 2 until all the v_i's are obtained.

2.5.3.4 Indexing objects by B$^+$-trees

In Figure 2.10, we describe a method to construct B$^+$-trees in a projected space, say M_i (others are addressed similarly). First, note that after the preceding steps, each mapped object is included in only one ring of the ring set $\{R_{j,l}\}$ in M_i, e.g., $o_1, o_2 \in R_{1,2}$ and $o_3 \in R_{2,1}$. For each ring $R_{j,l}$, we construct a B$^+$-tree $T_{j,l}$ to index all the objects in $R_{j,l}$. Let us focus on the ring $R_{1,2}$ and the corresponding vector v centered at C_1. We measure the angle of each object in $R_{1,2}$ to v as its key. Thus, the key of o_1 is $\pi/3$

Algorithm 3: Indexing phase of R2LSH

Input: $\mathcal{D}$ is the dataset; m is the number of projected spaces; α is the number of reference objects in a projection space; β is the number of rings centered at each reference object;

Output: the index structure $\{R^i_{j,l}, T^i_{j,l}\}$

```
Generate 2m hash functions {h_i}_{i=1}^{2m} by random projection;
Construct m two-dimensional projected spaces {M_i}_{i=1}^{m};
for i = 1 to m do
    ∀o ∈ D map o to M_i by o → (h_{2i-1}(o), h_{2i}(o));
    Select α reference objects {C_j^i}_{j=1}^{α} in M_i;
    Select reference vector v_i;
    for j = 1 to α do
        Construct β concentric rings {R_{j,l}^i}_{l=1}^{β} centered at C_j^i;
        for l = 1 to β do
            ∀o ∈ D compute o's reference angle to v_i;
            Construct B+-tree T_{j,l}^i;
Return {R_{j,l}^i, T_{j,l}^i} (1 ≤ i ≤ m, 1 ≤ j ≤ α, 1 ≤ l ≤ β);
```

and that of o_2 is $4\pi/3$. We can construct a B$^+$-tree $T_{1,2}$ with these keys. Note that in this manner, each object belongs to only one of the B$^+$-trees.

2.5.4 Query phase

In this section, we illustrate the query phase of R2LSH. The goal of this phase is to identify candidates by query-centric balls. In Sec. 2.5.4.1, by deriving certain fundamental relationships, we demonstrate a method to determine the radii of rings to ensure that o^*, the nearest neighbor of o, is identified with the expected success probability. In Sec. 2.5.4.2, we verify that all the objects inside the query-centric balls are necessarily scanned by our index structure. After the descriptions of these preparations, we present the algorithm and the related probability guarantee, in Secs. 2.5.4.3 and 2.5.4.4.

2.5.4.1 Fundamental relationships

First, let us consider a simper problem. Given query q and error rate $0 < \delta < 1$, when $s^* = \|o^* - q\|$ is known before the search, how do we identify o^* with probability at

least $1-\delta$? According to Equation (2.3), we have the following equation:

$$\Pr[\Delta_i(o^*) \leq r] = \Phi(\frac{r^2}{(s^*)^2}), \tag{2.13}$$

where r is a positive number, i ($1 \leq i \leq m$) is an arbitrary suffix of the projected space, and Φ is the CDF of the chi-square distribution $\chi^2(2)$. Then, according to the definition of $\Delta_i(o)$, Equation (2.13) is re-expressed as

$$\Pr[g_i(o) \in B(g_i(q), r)] = \Phi(r^2/(s^*)^2). \tag{2.14}$$

Meanwhile, from the fact that $\{M_i\}_{i=1}^m$ are constructed independently, we know that $\#Col(o^*; r)$ (the number of those projected spaces where o^* falls inside the query-centric balls with radii r) follows the Bernoulli distribution $Ber(m, \Phi(r^2/(s^*)^2)$. Therefore, by the Hoeffding's inequality, we have

$$\Pr[\#Col(o^*; r) \geq \tau] \geq 1 - \exp\{-2m(\Phi(r^2/(s^*)^2) - \tau/m)^2\} \tag{2.15}$$

where the positive integer τ denotes a threshold. This will be determined subsequently. For a specified error rate δ, to permit $\Pr[\#Col(o^*; r) \geq \tau] \geq 1-\delta$ to hold, we only require that the following inequality holds:

$$\begin{aligned} \Phi(\frac{r^2}{(s^*)^2}) &\geq \sqrt{\frac{1}{2m}\log\frac{1}{\delta}} + \frac{\tau}{m} \\ &= \lambda(m, \delta, \tau) \end{aligned} \tag{2.16}$$

When m, δ, and τ are evident from the context, we use λ for $\lambda(m, \delta, \tau)$. Because $\Phi(x)$ is monotonically increasing in x and a bijection from R^+ to $[0, 1]$, it suffices to adopt an r such that

$$r \geq r^* = \sqrt{(s^*)^2 \Phi^{-1}(\lambda)}. \tag{2.17}$$

Under Inequality (2.17), we can guarantee that o^* passes the collision testing, i.e. $\#Col(o^*; r) \geq \tau$, with probability at least $1-\delta$. Apparently, all the objects that pass this collision testing are considered as candidates. As the input parameters λ, m, and δ are specified, the collision threshold τ is determined from Equation (2.16) by

$$\tau = \left\lfloor m\lambda - \sqrt{\frac{m}{2}\ln\frac{1}{\delta}} \right\rfloor. \tag{2.18}$$

However, in practice, we cannot directly determine r^* by (2.17) because s^* is unknown. However, this challenge can be overcome by the dynamic counting technique proposed in [20]. At the beginning of the search, we set a query-centric ball of radius zero in each M_i. As the search proceeds, we increase the radius r of each ball in steps of Δr and update the collision numbers of data objects. During this process, we keep the nearest neighbor $\hat{o}$ of q among the accessed candidates and accurately calculate its distance to q, i.e., $\hat{s} = \|\hat{o} - q\|$. If we identify that $r \geq \hat{s}\sqrt{\Phi^{-1}(\lambda)}$ holds for a fixed λ, then we know $r \geq r^*$ from (2.17). This is because

$$r \geq \hat{s}\sqrt{\Phi^{-1}(\lambda)} \Rightarrow r \geq s^*\sqrt{\Phi^{-1}(\lambda)} \Rightarrow r \geq r^*. \tag{2.19}$$

Thus, $r > \hat{s}\sqrt{\Phi^{-1}(\lambda)}$ is considered as the termination condition of R2LSH.

2.5.4.2 Scanning Process

The remaining problem is to identify the objects that lie in query-centric balls. Because all the objects are contained in one of the rings in each projected space, we need to consider only the relationship between the query-centric ball and a ring. Given a query ball $B(g_i(q), r)$ and a ring region R centered at C with inner radius $\underline{r}$ and outer radius $\bar{r}$ in M_i. Let $s_0 = \|C - g_i(q)\|$, we consider the following two cases:

Case 1: $s_0 - r > \bar{r}$ or $s_0 + r < \underline{r}$. Evidently, in this case, the ball does not intersect with R. Therefore, we omit the examination of R.

Case 2: $\underline{r} - r \leq s_0 \leq r + \bar{r}$. In this case, we first compute the reference angle θ_q of $g_i(q)$ to the reference vector v_i of R. Then, we consider three more detailed cases and compute θ as follows:

$$\theta = \begin{cases} \arccos\left(\frac{(s_0)^2+\bar{r}^2-r^2}{2s_0\bar{r}}\right) & (s_0)^2 - r^2 > \bar{r}^2 \\ \arcsin(r/s_0) & \underline{r}^2 \leq (s_0)^2 - r^2 \leq \bar{r}^2 \\ \arccos\left(\frac{(s_0)^2+\underline{r}^2-r^2}{2s_0\underline{r}}\right) & (s_0)^2 - r^2 < \underline{r}^2 \end{cases} \tag{2.20}$$

It is evident from elementary calculation that given $r > 0$, we only need to scan those objects in R whose reference angles to v_i lie in $[\theta_q - \theta, \theta_q + \theta]$ so that all the objects that fall in $B(g_i(q), r)$ can be covered. It is convenient to access these objects in the external memory because they are sequentially stored (in mod 2π) in the leaf nodes of T, i.e., the corresponding B$^+$-tree of R. Therefore, the complete scanning process of

R can be illustrated as follows (Figure 2.10): given a query q, we first determine the location of θ_q in the leaf nodes of T. In each step, according to r, we can determine the search interval $[\theta_q - \theta, \theta_q + \theta]$ in which the objects need to be scanned. In the next step, r is increased by Δr, and θ is updated according to Equation (2.20). At this stage, we only need to extend $[\theta_q - \theta, \theta_q + \theta]$ in two directions. This can be realized conveniently at leaf levels of B^+-trees. If $\theta_q - \theta < 0$ or $\theta_q + \theta > 2\pi$, we only need to replace $\theta_q - \theta$ by $\theta_q - \theta + 2\pi$ or replace $\theta_q + \theta$ by $\theta_q + \theta - 2\pi$. This implies that the interval extends from the other end of the leaf level.

It is notable that for a few objects that lie outside balls, they are also likely to lie in the interval that needs to be examined (such as o_4 in Figure 2.10). However, for the desired object o^*, this only marginally increases the likelihood that it passes the collision testing. Therefore, the probability guarantee obtained in Sec. 2.5.4.1 still holds.

Finally, the setting of Δr can be computed by a minimum heap. Specifically, we only scan a leaf node of a B^+-tree in each step. According to the key value θ of the last scanned object in this step and (2.20), we can reversely compute r. Thus, for each B^+-tree that needs to be examined, we can maintain two corresponding values r_0 and r_1. Here, r_0 and r_1 correspond to the left and right directions, respectively. By means of the minimum-heap, in each step, we can determine the B^+-tree and the search direction corresponding to the minimum r_0 or r_1, and examine the next leaf node.

2.5.4.3 Algorithm and Quality guarantee

The query phase is displayed as Algorithm 4 for solving the c-ANN problem ($c \geq 1, k = 1$). The probability guarantee of Algorithm 4 is as follows:

Theorem 4. *For a specific query q and approximation ratio ($c \geq 1$), Algorithm 4 returns a c-approximate NN of q with probability at least $1 - \delta$.*

Proof. Let us consider the following two cases in order:

Case 1: ($c = 1$). According to the termination condition, the query phase can terminate only when $r \geq \sqrt{\Phi^{-1}(\lambda)}\hat{s}$ holds. By the preceding analysis, o^* is returned with probability at least $1 - \delta$.

Case 2: ($c > 1$). First, three events need to be defined. E_0: $\hat{o}$ is a c-approximate object, E_1: $\#Col(o^*; r^*) \geq \tau$, and E_2: $r < r^*$. We need to establish that $\Pr[E_0] \geq 1 - \delta$.

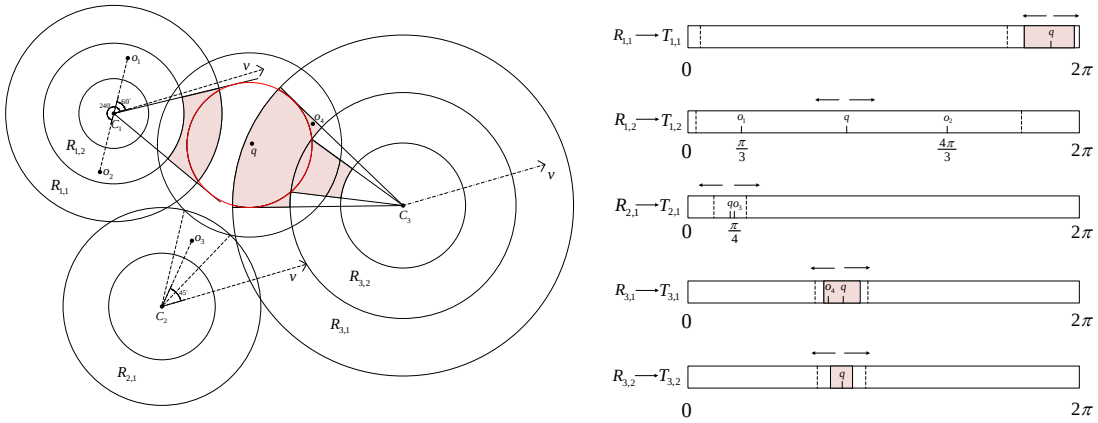


Figure 2.10: Example of B+-trees for three clusters in a projected space (Left: geometrical relationship. Right: leaf levels of B+-trees).

When E_2 occurs, we obtain $\hat{s} \leq cs^*$ according to the definition of E_2. This implies that we have identified c-approximate objects, i.e., E_0 occurs. When both $\bar{E}_2$ and E_1 occur, we also know E_0 occurs for o^* is found. Because E_2 and $E_1 \cap \bar{E}_2$ are disjoint, we obtain the following inequalities:

$$\Pr[E_0] \geq \Pr[E_2] + \Pr[E_1\bar{E}_2] \tag{2.21}$$

$$\geq \Pr[E_1E_2] + \Pr[E_1\bar{E}_2] \tag{2.22}$$

$$= \Pr[E_1] \tag{2.23}$$

Because $\Pr[E_1]$ has been established to be at least $1 - \delta$ in Case 1, we conclude.

□

2.5.4.4 Extension to c-k-ANN search

Algorithm 4 can be modified conveniently as follows to support c-k-ANN search:

(i) A priority queue Q needs to be maintained to store k nearest neighbors $\hat{o}_1, \ldots, \hat{o}_k$ among the candidates.

(ii) The termination condition in Step 3 needs to be modified as $r \leq \sqrt{\Phi^{-1}(\lambda)}\hat{s}_k/c$. Here, $\hat{s}_k = \|\hat{o}_k - q\|$.

Algorithm 4: Query phase of R2LSH

Input: $\mathcal{D}$ is the dataset; q is the query point; m is the number of projected spaces; $\{R_j^i, T_j^i\}$ $(1 \leq i \leq m,\ 1 \leq j \leq N)$ is the index structure, where N is the number of rings in a projected space; δ $(0 < \delta < 1)$ is the error rate; λ $(0 < \lambda < 1)$ is the parameter for computing the threshold; c $(c \geq 1)$ is the approximation ratio;

Output: $o_{\min}$

```
r ← 0, ŝ ← ∞, o_min ← NULL;
τ ← ⌊mλ − √(m ln(1/δ)/2)⌋;
while r ≤ √(Φ^{-1}(λ)) ŝ/c do
    r ← r + Δr (Δr > 0);
    for i = 1 to m do
        for j = 1 to N do
            if B(g_i(q), r) ∩ R_j^i ≠ ∅ then
                Update the search interval in T_j^i by (2.20);
            else
                continue;
    ∀o ∈ D update #Col(o; r) when necessary;
    if o is not visited and #Col(o; r) ≥ τ then
        Calculate ŝ = ‖q − o‖;
        Update o_min and ŝ when necessary;
Return o_min
```

When we apply our previous analysis on o^* to o_i^* $(1 \leq i \leq k)$ (where o_i^* is the true ith nearest neighbor of q), in accordance with Theorem 4, we have the following corollary:

Corollary 1. *For an arbitrary suffix i $(1 \leq i \leq k)$, the probability that a c-approximate nearest neighbor of o_i^* is returned by Algorithm 4 is at least $1 - \delta$. Moreover, if c is set to one, the expectation of recall rate is at least $1 - \delta$.*

2.5.5 Discussion

2.5.5.1 Complexity Analysis

Time Complexity. In the indexing phase, R2LSH consumes $O(nd)$ time to compute the images of each object in the projected spaces. Then, R2LSH consumes $O(n)$ time to partition the spaces based on reference objects and to select reference vectors. For

each ring, R2LSH consumes $O(n)$ time to compute the reference angle of each object to the reference vector and $O(n \log n)$ time to construct the corresponding B$^+$-tree. Therefore, the time complexity of the indexing phase is $O(nd + n \log n)$.

In the query phase, R2LSH requires $O(n)$ time to scan all the B$^+$-trees and $O(nd)$ time to accurately compute the distances between q and the candidates. This implies that in the worst case, the time complexity of R2LSH is linear. This does not imply that R2LSH is inefficient because unlike many other c-ANN methods, R2LSH can support any value of c (≥ 1), with probability guarantee. To the authors' knowledge, there is no search method can achieve the same goal in sub-linear complexity. On real datasets, R2LSH only computes the exact distances of q to a very small number of objects by the tuning of c and thus performs highly effectively, as demonstrated by experiments described in Section 2.5.6.

Space Complexity. In the indexing phase, R2LSH first consumes $O(md)$ space to restore m projection vectors. Then, R2LSH consumes the corresponding B$^+$-tree for each ring, consuming $O(n)$ space to restore the projections of the objects. During the phase of constructing B$^+$-trees, R2LSH requires a buffer of size equal to the I/O page size, to restore a leaf/index node of the B$^+$-tree. Therefore, the total space cost is $O(n + md)$.

In the query phase, before the search, R2LSH first needs to load certain auxiliary information of the index structure into the main memory. The information includes those on (1) $2m$ projection vectors, (2) reference objects and the reference vector in each M_i, and (3) the inner and outer radii of each ring, which is totally $O(md)$. During the search process, R2LSH needs to maintain an array of size n to record the present collision numbers of the objects. In addition, for each ring that needs to be searched, R2LSH maintains two buffers to restore the I/O pages read from disks. This is because the leaf level is searched in two directions. If the I/O page size is set to 4KB, the space cost of this part is at most $8m\alpha\beta$ KB. α and β are defined in Table 2.3. Because $n \gg m\alpha\beta$ for large datasets, the total space cost in the query phase is also $O(n + md)$.

2.5.5.2 Handling Update and parameter setting

Update. For the datasets updated frequently, we recommend users to adopt a simple selection strategy. This is because the learning step in the indexing phase is not

required, implying that the cost of re-learning owing to the alterations in data distributions could be saved. In the experiments, we demonstrate that for most datasets, a simple selection strategy can be effective. Meanwhile, because our index structure is formed by B^+-trees, the update (insertion, deletion, and modification) can be supported conveniently.

Parameter setting. The parameters that need to be determined are m, λ, and β. It is evident that their optimal values depend on both data distribution and query distribution. To support dynamic and large datasets, in this study, we experimentally determined the default values of these three parameters and applied these values to all the real datasets. The experiments revealed that R2LSH could perform effectively under such default setting of parameters.

2.5.6 EXPERIMENTS

R2LSH was implemented in C++. All the experiments were carried on a PC with Intel(R), 3.40 GHz i7-4770 eight-core processor with 8 GB RAM, in Ubuntu 16.04, unless stated otherwise.

2.5.6.1 Experiment Setup

We chose following methods in the experiments.

- **SRS [51].** We select SRS-2, a variant of SRS, in our experiments. This is because SRS-2 supports arbitrary $c \geq 1$ and $0 < \delta < 1$. The dimension of the R-tree was set to eight, as recommended in [25, 35, 51]. δ was set to 0.1.
- **QALSH [26].** According to the authors' recommendations, δ was set to $1/2 + 1/e$. The number of hash functions m and other internal parameters were computed according to the methods proposed in [26] as well.
- **R2LSH.** δ was set to 0.1. The parameters λ, m, and β were set as described in Sec. 2.5.6.1. In addition, to compare R2LSH and the other LSH methods without bias, we adopted the simple selection strategy ($\alpha = 1$) introduced in Sec. 2.5.3.2 (unless stated otherwise). This causes all LSH methods to be learning-free.
- **HD-index [5].** We select HD-index as a representative of non-LSH methods (without quality guarantees). All the internal parameters of HD-index were adjusted to be experimentally optimal, as recommended in [5].

Table 2.4: Real datasets

Dataset	Dimension	Size	Type
Sun	512	79,106	Image
Enron	1369	94,987	Text
Msong	420	992,272	Audio
Deep	256	1,000,000	Image
Gist	960	1,000,000	Image
Imagenet	150	2,340,373	Image
Tiny10M	384	10,000,000	Image
Sift10M	128	11,164,666	Image
Tiny80M	384	79,302,017	Image
Sift1B	128	999,494,170	Image

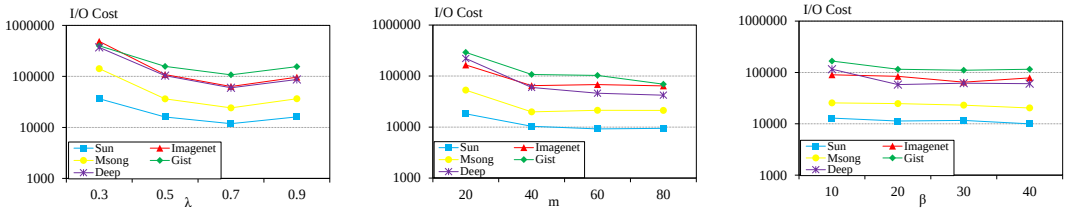
Datasets. We select ten real datasets of different sizes [5, 35, 51] (Table 3.3):

The I/O page sizes, denoted by B, were set to 4KB on all the datasets, as suggested in [5].

Evaluation metrics. We used the following metrics for the performance evaluation.

- **I/O cost.** I/O cost, which denotes the number of pages to be accessed, is an important metric for external memory algorithms. We follow SRS and QALSH to use this metric to evaluate the efficiency. As discussed earlier, I/O costs consist of the overhead for both index and data access.
- **Running time.** The running time for processing a query is also considered. It is defined as the wall clock time for a method to solve the k-ANN problem.
- **Overall ratio.** For methods to solve c-k-ANN search, overall ratio is a standard metric used to measure the query accuracy. For c-k-ANN search, the overall ratio is defined as $\frac{1}{k}\sum_{i=1}^{k} \|q-\hat{o}_i\| \,/\, \|q-o_i^*\|$.
- **Recall.** Recall is used as another important metric to measure the accuracy of algorithms. Its value is equal to the ratio of the number of returned true nearest neighbors to k (for k-ANN search).

Parameter settings. We recommend users to set λ, m, and β to 0.7, 40, and 30, respectively, by default. This is because such settings are suitable for most datasets. In Figure 2.11, we demonstrate the effect of different parameters on I/O costs, through experiments on five datasets.



(a) Different λ ($m = 40$, $\beta = 30$)

(b) Different m ($\lambda = 0.7$, $\beta = 30$)

(c) Different β ($\lambda = 0.7$, $m = 40$)

Figure 2.11: The effect of different parameters in R2LSH

- Figure 2.11(a) shows that $\lambda = 0.7$ is suitable because the least I/O cost is incurred on each dataset under this setting. If λ is set too small or too large, it will be challenging to distinguish NNs from other false objects owing to their close possibilities of falling inside query-centric balls.
- Figure 2.11(b) shows that m should not be too small because R2LSH requires an adequate number of balls for the collision testing to be efficient. Meanwhile, when $m \geq 40$, almost all the curves remain stable except the curve of Gist. This implies that it is effective to set m to 40. With regard to Gist, we can achieve higher performance when m is set larger because Gist is a high-dimensional dataset and requires more balls in the indexing phase.
- Figure 2.11(c) shows that the I/O cost required under $\beta \geq 20$ is marginally less than that under $\beta = 10$. This is because we can scan a lower number of objects by using more rings. Meanwhile, β should not be too large because we have to generate a B+-tree for each ring. Therefore, we set β to 30. This is observed to be suitable for all the five datasets.

Cluster-based selection vs Simple selection. On all the datasets of size within 1 M, we compared R2LSH with $\alpha = 4$ (four reference objects were determined by K-means) and the standard R2LSH ($\alpha = 1$) with the simple selection strategy, under $c = 1$. Compared with the standard R2LSH, R2LSH ($\alpha = 4$) can save $5\% \sim 20\%$ I/O cost on each dataset. This implies that R2LSH based on cluster-based selection performs marginally higher than the standard R2LSH even in the general case.

Table 2.5: B^+-tree vs. R-tree on running times(s) necessary to achieve overall ratio 1.01. @k means the number of nearest neighbors.

	Sift(@1)	Sift(@10)	Tiny(@1)	Tiny(@10)
B^+-tree	5.3	7.5	9.9	11.2
R-tree	23.2	33.3	46.3	60.7

Table 2.6: Ratios of I/O costs in index access of R2LSH to those of QALSH. @k means the number of nearest neighbors.

	Sift(@1)	Sift(@10)	Tiny(@1)	Tiny(@10)
Ratio	0.05	0.04	0.06	0.06

2.5.6.2 Efficiency of R2LSH

In this section, we describe the experiment by which it was demonstrated that the design of R2LSH is reasonable. It concerns the following two aspects: (1) the superiority of B^+-trees over other tree structures, and (2) the superiority of Dimension 2 (the subspace dimension of R2LSH) over other dimensions. All the experiments described in this section were performed on Tiny10M and Sift10M.

B^+-tree vs. R-tree. We first demonstrate that B^+-trees are superior to other tree structures in the searching process of R2LSH. Here, we select R-tree as the benchmark tree structure because it can be realized in the external memory. In this case, we adjust the number of subspaces and the dimension of R-trees for them to be experimentally optimal. Moreover, we use an R-tree to scan objects sequentially as the radius of the search ball increases in each subspace. The comparison results are presented in Table 2.5. The results reveal that R-trees are less effective than B^+-trees in our searching scheme. This is because objects are stored sequentially on B^+-trees, whereas the scanning on R-trees results in significantly more random I/Os (other low-dimensional tree-structures also exhibit this limitation). Thus, for our algorithm to save time, we use B^+-trees to form our data structure.

Dimension 2 vs. Dimension 1. Because the query-centric hash bucket can be considered as the one-dimensional query-centric ball, it suffices to demonstrate the superiority of R2LSH (Dim 2) over QALSH (dim 1). In this section, we focus only on the comparison results of I/O costs in index access, which reflects the most important advantage of R2LSH over QALSH. The results in Table 2.6 illustrate that R2LSH

Table 2.7: Running times(s) necessary to achieve overall ratio 1.01 under different subspace dimensions. @k means the number of nearest neighbors.

	Sift(@1)	Sift(@10)	Tiny(@1)	Tiny(@10)
Dim 2	5.3	7.5	9.9	11.2
Dim 3	13.2	18.5	26.9	30.8
Dim 4	15.4	22.7	32.3	35.6

Table 2.8: Index sizes ($c = 2$). CR denotes crash in the indexing phase.

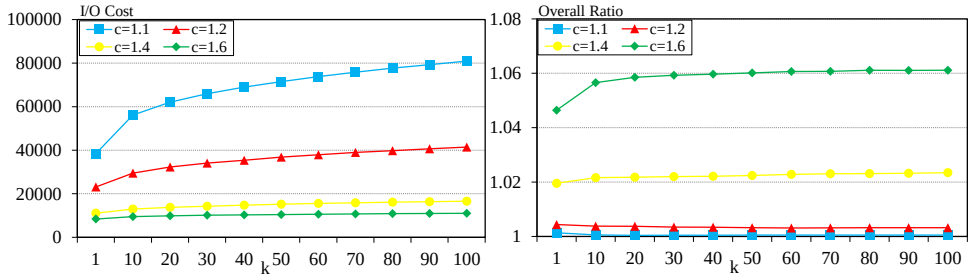
Dataset	SRS	QALSH	R2LSH	HD-index
Sun	4.1MB	21.2MB	13.5MB	250MB
Enron	4.6MB	26.5MB	16.6MB	1.5GB
Msong	47.5MB	341.1MB	164.4MB	3.9GB
Deep	47.4MB	350.9MB	169.1MB	2.0GB
Gist	47.3MB	350.9MB	169.1MB	18.1GB
Imagenet	110.8MB	836.0MB	380.0MB	4.1GB
Tiny10M	476.7MB	3.8GB	1.6GB	31.3GB
Sift10M	524.7MB	4.1GB	1.8GB	10.2GB
Tiny80M	3.6GB	CR	12.6GB	241.3GB
Sift1B	51.2GB	CR	177.8GB	1.2TB

requires 4%–6% of the I/O cost for index access of QALSH because of the higher selectivity of two-dimensional query-centric balls.

Dimension 2 vs. higher dimensions. R2LSH can be conveniently generalized to support higher subspace dimensions. Here, for each hyper-ring, we construct a B^+-tree to store the angles of objects to a reference vector (although we can construct multiple B^+-trees for each subspace, this will cause the total number of B^+-trees to be excessive) and search for the candidates similarly. Table 2.7 presents the results under different subspace dimensions. The results reveal that when we select a higher dimension ($>$ 2), the performance is inferior to that under Dimension 2. This is because it is more challenging to prune false objects in subspaces of higher dimensions unless more B^+-trees or other low-dimensional tree structures are used. However, as discussed earlier, this will also incur significant additional cost.

2.5.6.3 Index Size, Indexing Time, and Memory Consumption

Table 2.8 presents the index sizes of three methods. Because the index size of QALSH depends on the approximation ratio c, we report only the results under $c = 2$. According to the results, on each dataset, the index size of SRS is the smallest, followed by



(a) Gist, I/O Cost **(b)** Gist, Overall Ratio

Figure 2.12: I/O Cost and accuracy of R2LSH vs. c

R2LSH, QALSH, and HD-index. This is because the index structure of SRS is an R-tree, whereas R2LSH and QALSH use multiple B^+-trees. Meanwhile, although the number of B^+-trees of R2LSH (1200) is more than that of QALSH (60-100), each object appears only 40 (equal to the number of projected spaces) times in the index structure of R2LSH. Therefore, the index size of R2LSH can be less than that of QALSH.

With regard to the indexing time, on our PC, R2LSH is the fastest, followed by QALSH, SRS, and HD-index, on all the datasets. For example, on Sift1B, R2LSH, SRS, and HD-index consume 13 h, 5 days, and 11 days, respectively, to construct the index. The construction of B^+-trees is more time-consuming than that of an R-tree. Thereby, SRS consumes more time in the indexing phase although its index size is the smallest among the three LSH-methods.

In addition, we report the main memory consumption of each method on the largest dataset Sift1B as follows. In the indexing phase, SRS, R2LSH, and HD-index consume 1.1GB, 1.6GB, and 97MB. SRS and R2LSH are both projection-based methods and require more space in the main memory to store projection values.

2.5.6.4 The effect of c

In Figure 2.12, we study the effect of different values of the approximation ratio c on the performance of R2LSH. Owing to the space limitation, we present the results only on Gist. Similar trends were observed on other datasets. The following observations are based on the results:

- The real overall ratio is significantly smaller than the value of c, a required lower bound. This is because the probability guarantee of R2LSH is obtained in the worst case, and the performances of R2LSH on real datasets can be significantly higher.

Table 2.9: I/O costs ($\times 10^5$) necessary to achieve overall ratio 1.01 on Tiny80M and Sift1B. @k means the number of nearest neighbors.

	Sift(@10)	Sift(@100)	Tiny(@10)	Tiny(@100)
R2LSH	81	98	6	9
SRS	307	379	55	70

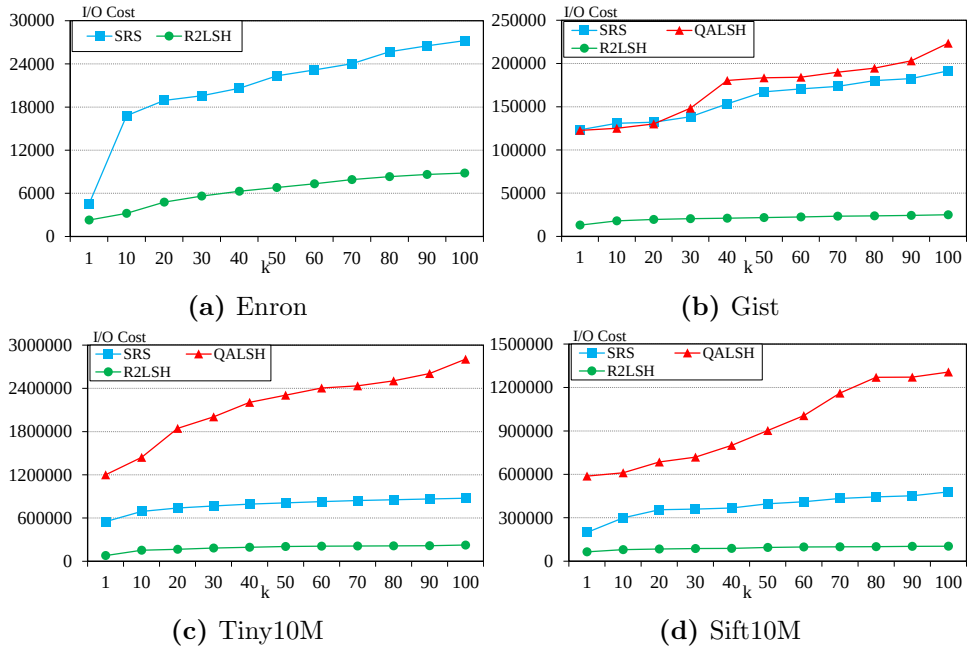


Figure 2.13: I/O costs necessary to achieve overall ratio 1.01

- As c decreases, R2LSH achieves higher query quality at the expense of higher I/O cost. This is consistent with our termination condition in Algorithm 2. This implies that according to the practical requirement, we can achieve a good balance between query cost and precision by tuning the approximation ratio. Figure 5 shows that R2LSH performs well under $c = 1.2$.

2.5.6.5 R2LSH vs. other LSH-based methods

This section describes our comparison of R2LSH with other state-of-the-art LSH-based methods, i.e., SRS and QALSH.

Comparison on I/O costs. First, we compare the required I/O costs of R2LSH, QALSH, and SRS (Figure 2.13 and Table 2.9). Please note that we measured I/O costs or time necessary for achieving a specified value of overall ratio instead of a same

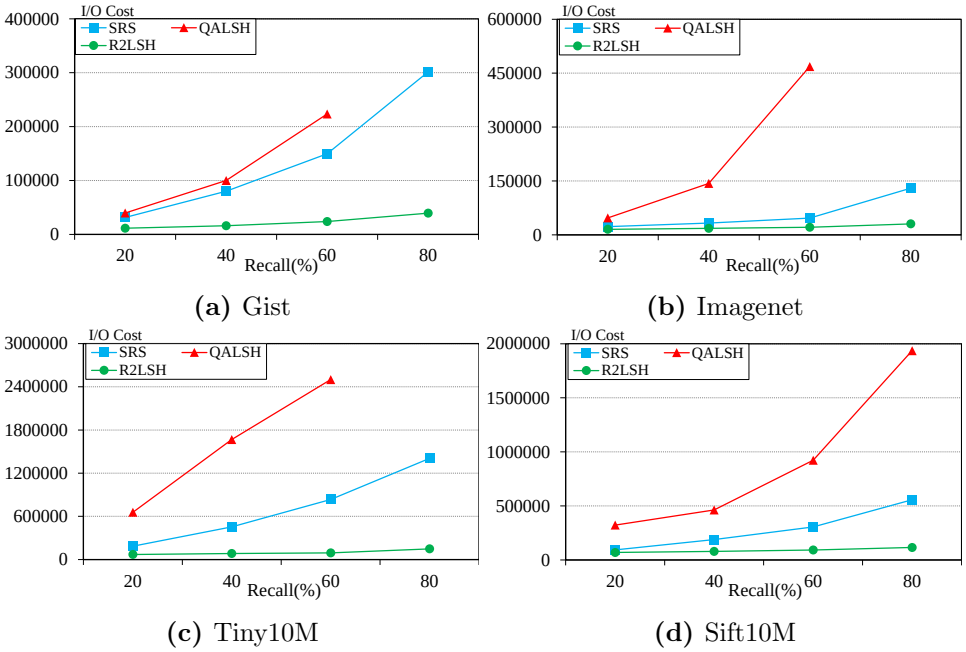


Figure 2.14: I/O costs given recalls 20%, 40%, 60% and 80% ($k = 100$)

value of approximation ratio c [1]. This is because R2LSH, QALSH, and SRS adopt different methods to obtain probability guarantees. Consequently, SRS incurs less I/O cost, whereas QALSH achieves higher accuracy for the same value of c. Note that all the three methods can support arbitrary c ($c > 1$) and can achieve different accuracies by tuning c. Therefore, for fair comparison, we set the overall ratio to a constant and measure the I/O cost they incur to attain the objective overall ratio by tuning c. In Figure 2.13, we fix the overall ratio to 1.01 and compare the performances of the three methods. $c = 1.01$ is selected because it is adequately low, and all the methods can achieve this ratio on most of the datasets. The results reveal that the performance of R2LSH is the highest on each dataset. Specifically, R2LSH incurs 15%–30% of the I/O costs incurred by SRS or QALSH to achieve an identical precision level. This reveals that R2LSH is more I/O efficient than other LSH-based methods. We also observe that the performance of QALSH degrades significantly as the data size increases. This is because QALSH first fixes the size of the candidate set and then computes the number of required hash functions. For large datasets, a similar setting makes the candidate set

[1] In previous papers, $c = 2$ is considered as the standard setting. However, it is impractical because for many high-dimensional real datasets, over 30% of the data objects are candidates when $c = 2$.

Table 2.10: Speeds(s) given recall 70% ($k = 100$). 70% is selected because it can be achieved by three methods and corresponds to targeted overall ratio of approximately 1.01.

Dataset	R2LSH	Gain of R2LSH over	
		QALSH	SRS
Sun	0.28	1.00x	**3.02x**
Enron	0.41	\	**2.97x**
Msong	0.93	**2.83x**	**4.28x**
Deep	1.41	**4.33x**	**5.63x**
Gist	2.70	**5.66x**	**3.70x**
Imagenet	2.53	**12.19x**	**5.72x**
Tiny10M	12.66	**14.61x**	**6.77x**
Sift10M	9.51	**9.15x**	**7.13x**
Tiny80M	81.53	\	**7.82x**
Sift1B	793.18	\	**7.27x**

of QALSH too small whereas the I/O cost of index access becomes excessively large. This makes the search inefficient.

In Figure 2.14, we vary the targeted recall and compare these methods under $k = 100$. We can see that under different precision levels, R2LSH always exhibits the highest performance. In addition, as the targeted recall increases, the advantage of R2LSH over SRS or QALSH becomes more obvious.

Comparison on running times. We also compare the running times of the three methods in Table 2.10. We first observe that R2LSH exhibits apparent advantages over SRS on speeds. This is because the index structure of R2LSH is formed by B^+-trees. During the scanning process, the objects on B^+-trees can be scanned sequentially, which is efficient in terms of time-consumption. In contrast, SRS processes objects by an R-tree, which incurs more time for distance computation and searching for new index nodes.

Meanwhile, we observe that R2LSH runs significantly faster than QALSH, particularly on large datasets. This is consistent with the results on I/O costs. It is noteworthy that on the small dataset Sun, the running times of two methods are very close. This is because R2LSH constructs more B^+-trees than QALSH on small datasets, which incurs more time for locating projections of the query. However, such cost is comparatively small for large datasets.

Chapter 3

Maximum Inner Product Search Problem

3.1 Introduction

In this chapter, we focus on the inner product space and research the *Maximum inner product search problem(MIPS)*, which has been regarded as a fundamental problem in many applications such as recommender systems [34], multi-label predictions [18], reasoning concerning extracted facts in open relation extractions [45], deep learning [50], and structural SVM [30]. Given a large dataset $\mathcal{D}$ in $\mathbb{R}^d$ with an L_2 norm and a query q, the objective of MIPS is to find the MIP object o in $\mathcal{D}$ such that the inner product $\langle o, q \rangle$ is the maximum. We call such a problem the *exact MIPS problem*. Although the objective is straightforward, an exact MIPS problem is difficult to solve efficiently in high-dimensional spaces, owing to the curse of dimensionality. Hence, researchers have focused on the *approximate MIPS problem*, for which approximate MIP objects are acceptable, and proposed various approximate MIPS methods. For some of these approximate methods, the approximation ratio c is introduced to control the difference between the true and approximate MIP objects. Specifically, an approximate MIP object is called a c-MIP object (of q) if its inner product with q is not less than c times the true maximum inner product; we refer to those approximate MIPS methods that can return c-MIP objects as *c-approximate MIPS methods*.

In recent years, many exact and approximate MIPS methods have been proposed. They can be classified into the following three classes:

(1) **Methods that exploit suitable index structures to realize an efficient**

search. They exploit special data structures, such as balls and trees [16, 44]. In general, they perform well in low-dimensional spaces. But they are defeated by the brute-force search in high-dimensional spaces owing to the curse of dimensionality.

(2) **Methods that transform the MIPS problem into approximate nearest neighbor search (ANNS) or maximum cosine similarity (MCS) problems**. This class includes L2-ALSH [48], Sign-ALSH [49], Simple-ALSH [43], XBOX [6] and H2-ALSH [27]. Owing to the utilization of the locality sensitive hashing (LSH) technique, these methods can return c-MIP objects with probability guarantees and are efficient. However, in the probability guarantees, there exists a significant gap between theory and application.

(3) **Methods that generate data-dependent binary codes, typically by deep leaning**. The representative method is asymmetric inner-product binary coding (AIBC) [47]. Compared with data-independent methods in the second class, these methods can generate more efficient codes by fully exploiting data distributions. However, their learning are more time consuming, which presents a problem in feasibility on extremely large or dynamic datasets. Furthermore, owing to the lack of probability guarantees, it is difficult to improve their query accuracies to user-specified levels.

3.2 Related work

Because MIPS is a fundamental problem, it has gained significant attention from researchers. In this section, we briefly review some state-of-the-art MIPS methods.

3.2.1 Exact MIPS methods

The most straightforward solution is brute-force searching, i.e., to compute the inner products between the query vector and the vectors, and output all probe vectors with the largest inner product. However, it is costly in practice especially for large and high-dimensional datasets. Hence, other exact MIPS methods have been proposed. A practicable solution is to use tree-based index structures for searching, such as cone trees [44] and cover trees [16]. Another efficient method for an exact MIPS is LEMP [53], which prunes false objects quickly based on an efficient index structure. In general, these methods yield good performances on low-dimensional datasets; however, as the dimension increases, their effectiveness is lost because of the curse of dimensionality.

3.2.2 LSH-based MIPS methods (learning-free)

In many applications, approximate MIP objects can be accepted by users and found more efficiently than exact MIP objects. Thus, many approximate MIPS methods have been proposed to trade accuracy for speed. To overcome the curse of dimensionality, these methods first transform MIPS to ANNS or MCS and then adopt LSH techniques (by random projection) to solve the transformed problems. Owing to the favorable properties of LSH, such methods can perform well even for high-dimensional datasets. According to the type of transformed problem, these methods can be divided into the following two categories. 1) Methods in the first category transform MIPS to ANNS, such as L2-ALSH [48] and H2-ALSH [27]. After the transformation, L2-ALSH solves ANNS by E2LSH [1], while H2-ALSH solves ANNS by QALSH [26]. 2) Methods in the second category transform MIPS to MCS, such as Sign-ALSH [49] and Simple-LSH [43]. After the transformation, Sign-ALSH and Simple-LSH solve MCS by SimHash [13].

For existing methods of such a type, an efficient transformation is paramount under the condition that two objects with a large inner product are close to each other in the mapped space. The transformations are further divided into symmetric and asymmetric transformations. If the same transformation applied to probe vectors is applied to the query, we call it a *symmetric transformation*. Otherwise, we call it an *asymmetric transformation*. Simple-LSH adopts the symmetric transformation, while L2-ALSH, Sign-ALSH, and H2-ALSH adopt the asymmetric transformation.

Although such transformations render the utilization of LSH techniques feasible, existing LSH-based methods exhibit the following limitations. 1) For some methods, such as L2-ALSH and Sign-ALSH, *the transformation error* cannot be avoided. That is, the order of the inner products cannot be perfectly preserved after converting MIPS to ANNS/MCS, which results in inaccurate query results. 2) For other methods such as XBOX and H2-ALSH, the transformation error can be avoided but another type of error, e.g., *the distortion error* [27], cannot be avoided. Specifically, two objects with a large inner product may become very close to each other in the mapped space owing to the transformation, which increases the searching difficulty. Although it has been experimentally shown that H2-ALSH can incur fewer distortion errors than other methods, such an error is inevitable.

Table 3.1: Comparison among state-of-the-art MIPS methods

Methods	Approximate or exact solver?	Learning method?	Probability guarantee?	Transformation required?
Sign-ALSH [49]	Approximate	No	Yes	Yes
Simple-ALSH [43]	Approximate	No	Yes	Yes
H2-ALSH [27]	Approximate	No	Yes	Yes
AdaLSH(proposed)	Both	No	Yes	No
AIBC [47]	Approximate	Yes	No	No

3.2.3 Learning MIPS methods

In recent years, many learning-based methods have been proposed. By fully exploiting the data distribution, those methods generate efficient binary codes. However, as indicated in [47], most learning-based methods were designed to address approximate nearest neighbor search and their performances in solving the MIPS problem are not highly satisfactory. Hence, the authors of [47] developed an asymmetric binary code learning method called AIBC for solving MIPS problems. AIBC learns two sets of coding functions such that the inner products between their generated binary codes can reveal the inner products between the original data, which shows a good accuracy-efficiency tradeoff in practice.

3.3 Preliminaries

Before we provide the details of AdaLSH, we will provide some introductions in this section. In Sec. 3.3.1, we briefly review the mechanism of a highly related method, H2-ALSH and analyze its limitations. In Sec. 3.3.2, we introduce some notations and definitions for later use.

3.3.1 Brief review of H2-ALSH

The recently proposed H2-ALSH [27] outperforms other state-of-the-art methods of LSH type. In the preprocessing phase, H2-ALSH partitions a dataset into disjoint subsets $S_1, S_2, \ldots, S_n$, which can be viewed as concentric rings geometrically (S_1 has the smallest radius, followed by S_2, and so on). Subsequently, in each subset S_i, objects (in S_i) are mapped into a one-dimension-higher $(d+1)$ space, such that an object with a larger value of inner product with q in the original space will have a smaller distance

Table 3.2: Some notations

Notations	Explanations
δ	The error rate specified by users ($0 < \delta < 1$)
c	The approximation ratio specified by users ($0 < c \leq 1$)
o^*	The true MIP object of q
$\hat{o}$	The MIP object among candidates
n	The number of rings
S_i	The ith ring with inner radius r_i and outer radius r_{i-1}
b	$b = r_i/r_{i-1}$ for all i's
m	The number of hash functions
a_i	The ith projection vector
h_{a_i}	The ith hash function generated by random projection
$[-w_i, w_i]$	The search window in S_i
$\#Col(o_1, o_2)$	The collision number of objects o_1 and o_2
t_j^i	The threshold for w_j at the ith round

to q in the mapped space. In the query phase, from S_n to S_1, H2-ALSH repeats the ANN search.

Although H2-ALSH performs well, it exhibits the following two limitations. (1) Its probability guarantee is not always based on correct assumptions on data distributions[1]. In other words, for some specific data distributions, the real performances of H2-ALSH may be inconsistent with the predicted performances. (2) As mentioned in Sec. 3.1, the success probability guaranteed by H2-ALSH is low (only $1/2 - 1/e$). Therefore, to fulfill the user-specified precision, a number of repetitions are required to increase the success probability, which incurs additional cost.

3.3.2 Notations and problem setting

We consider a set $D \subseteq \mathbb{R}^d$ of objects o's and a query $q \in \mathbb{R}^d$. Our goal is to find $o^* \in D$, whose inner product with q is maximal. Let the angle between o and q be θ ($0 \leq \theta \leq \pi$). We use the tilde ($\sim$) for normalization, i.e., $\tilde{o} = o/\|o\|$, and the hat ($\wedge$) for "optimal candidates," such as $\hat{o}$ for o^*. The goal of the c-approximate maximum inner product (c-AMIP) problem is to find object $\hat{o}$ satisfying $\langle \hat{o}, q \rangle \geq c\langle o^*, q \rangle$ for a given c ($0 < c \leq 1$) ($c = 1$ for an exact MIPS). Herein, we primarily consider the method to find a single c-AMIP object; however, it is easy to extend the algorithm to

[1]In Theorem 4.2 [27], they limit the cosine angle in $[a, 1]$ for some $a > 0$, specifically $a = c_0^4 - 1/(bc_0^4)$ for parameters c_0 and b; however, this does not always hold. In addition, for an arbitrary object o, the angle between o and q is assumed to be uniformly distributed in $[0, \pi]$, which is impractical.

find k c-AMIP objects (we only need to ensure that the first MIP object corresponds to the kth one). In addition, if $\langle o^*, q\rangle$ is negative, a slight modification is required in our algorithm, such as replacing c by $1/c$. As the modification is effortless, we only focus on the typical case $\langle o^*, q\rangle > 0$.

Table 3.2 lists some important notations herein for reference. Some of them will be introduced later.

3.4 Adaptive LSH

In this section, we explain how AdaLSH works.

3.4.1 The indexing phase

First, AdaLSH generates m projection vectors $\{a_j\}_{j=1}^{m}$. Next, AdaLSH divides the dataset D into n distinct subsets $S_1, S_2, \ldots, S_n$ that are separated by concentric circles centered at zero with radii $r_0, r_1, \ldots, r_n$ in the descending order ($\max_{o\in D} \|o\| = r_0 > r_1 > \cdots > r_n > 0$), that is, $S_i = \{o | r_i < \|o\| \leq r_{i-1}\}$. Let $\ell_i = \min_{o\in S_i} \|o\|$, and $u_i = \max_{o\in S_i} \|o\|$. Subsequently, AdaLSH normalizes objects falling inside the same ring and projects them onto the generated vectors by hash functions $h_{a_i}(\tilde{o}) = \langle a_i, \tilde{o}\rangle$, where $\tilde{o}$ is the normalized object in the ring. We apply such an operation to all rings and thus obtain n groups of hash functions, each of which corresponds to some ring (See Figure 3.1 for an example).

3.4.2 The query phase

This phase is the core of AdaLSH. First, we present an overview of our search strategy. Next, we derive some basics and fundamental relationships. Subsequently, we introduce our search strategy in detail. Finally, we present the algorithm and describe the related probability guarantee. For clarity, we primarily address the case $c = 1$ (exact MIPS) and generalize it for $c < 1$ later.

3.4.2.1 Overview

The search process was performed over multiple rounds. In each round, we examined $S_1, S_2, \ldots, S_n$ from the outside to the inside. That is, we first limited the objects by their norms. In each subset S_i whose member o has a norm in $[\ell_i, u_i]$, we further limited

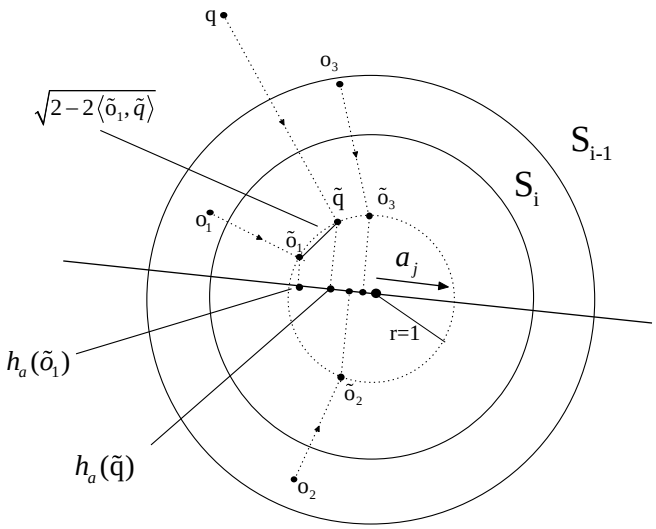


Figure 3.1: Indexing phase of AdaLSH. Any object o is normalized to $\tilde{o}$ ($\|\tilde{o}\| = 1$) and then projected onto a line with a random Gaussian vector a_j for collision testing.

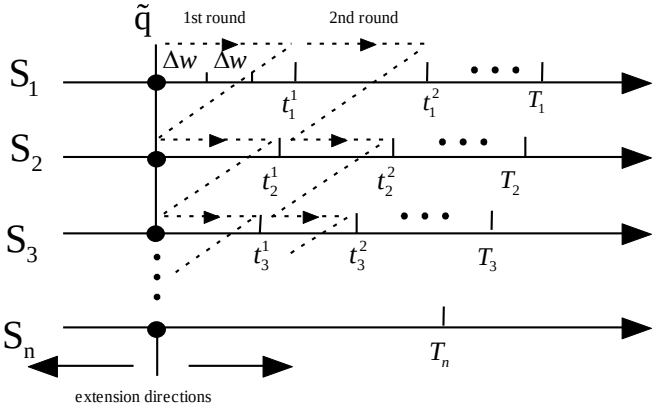
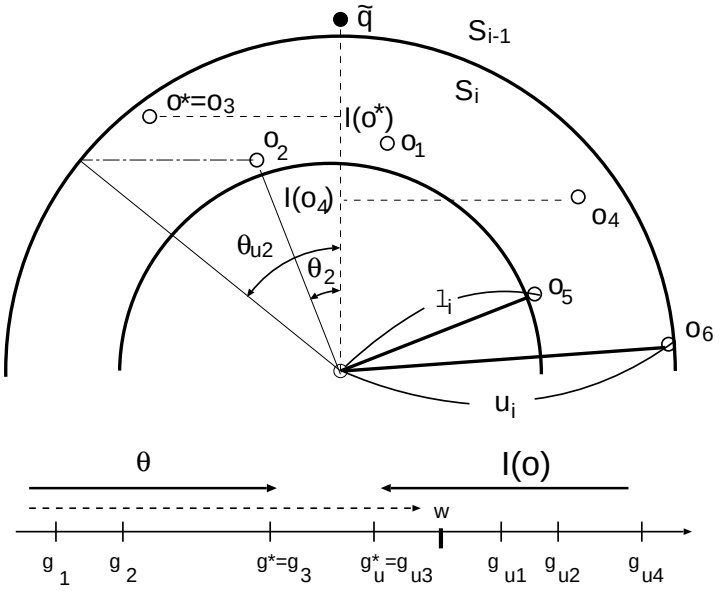


Figure 3.2: Multi-round search strategy, where t_i^r denotes the threshold for window size w_i at rth round



$$g_j = g(\cos\theta_j; p_0) = g(I(o_j)/\|o_j\|; p_0)$$
$$\leq g_{uj} = g(\cos\theta_{uj}; p_0) = g(I(o_j)/u_i; p_0)$$

Figure 3.3: Probabilistic examination order of norm-limited objects. All objects $o \in S_i$ are limited in their norm as $\ell_i \leq \|o\| \leq u_i$. Function g maps $\cos\theta = I(o)/\|o\|$ with $\tilde{q}$ to $c\sqrt{2-2\cos\theta}$ (c is a constant), where $I(o) = \langle o, \tilde{q}\rangle$ and $\|\tilde{q}\| = 1$. The objects are examined in the ascending order θ as $g_1, g_2, \ldots$, while the corresponding $g_{u_1}, g_{u_2}, \ldots$ are examined if $g_{u_j} < w$. Let $w = g(\hat{I}/u_i)$ with the current maximum inner product $\hat{I}$. Subsequently, if $I(o_j) > \hat{I}$, then $g_{u_j} < w$; accordingly, $g_j < w$. Therefore, it is sufficient to verify objects satisfying $g_j < w$.

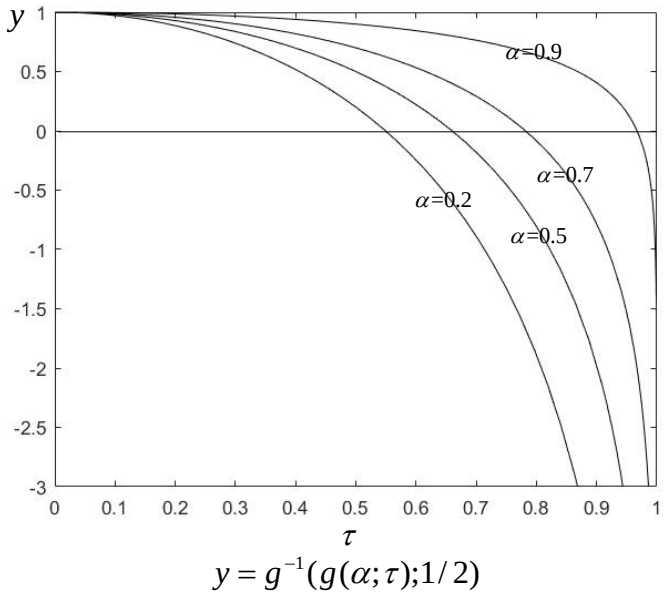


Figure 3.4: Functions for setting round values

the objects by their cosine angles $\cos\theta$, which were estimated correctly to support the query-centric LSH technique. The cosine angle was estimated through collision testing in multiple hash functions (projection lines), which will be explained later in detail, with a random vector $a \sim N(0, I_d)$, $h_a(\tilde{o}) - h_a(\tilde{q}) \sim N(0, 2-2\cos\theta)$ (Figure 3.1). That is, $\tilde{o}$ and $\tilde{q}$ have a high collision probability in a window of fixed size $2w$ if they are close to each other, i.e., $\cos\theta \approx 1$, while they have a low collision probability if they are distant from each other, i.e., $\cos\theta \approx -1$. Based on this property, we estimated the value of $\cos\theta$ by the collision number in m projection lines associated to randomly generated $a_1, \cdots, a_m$. We established an anchor window $[-w_i, w_i]$ in common to m projection lines for the collision tests of $o \in S_i$ and q. For verifying the objects in the descending order of the cosine angles, the value of w_i was increased gradually by Δw. When w_i became larger than a round threshold t_i^r in the rth round, we terminated the search in S_i and proceeded to verify the next subset S_{i+1} (Figure 3.2). When S_n has been processed, the next round starts. The details will be explained in Sec. 3.4.2.4.

3.4.2.2 Basics

Let us start by summarizing the facts that will be applied in this study. The inner product is defined as

$$\langle o, q \rangle = \|o\| \|q\| \cos\theta. \tag{3.1}$$

The solution of the problem, MIP$(\mathcal{D}, q)$, is given as

$$o^* = \arg\max_{o \in \mathcal{D}} \|o\| \|q\| \cos\theta \tag{3.2}$$

$$= \arg\max_{o \in \mathcal{D}} I(o) (= \|o\| \cos\theta). \tag{3.3}$$

It is noteworthy that the inner product $I(o)$ is evaluated as the product of the norm of o and the cosine angle with the query.

Let us consider a random hash function $h_a(o) = \langle a, o \rangle$, where a is generated randomly according to the d-dimensional standard Gaussian distribution $\mathcal{N}(0, I_d)$. It is well known [17] that for two points o and q,

$$h_a(o - q) = h_a(o) - h_a(q) \sim \mathcal{N}(0, \|o - q\|^2). \tag{3.4}$$

Suppose that o and q have been normalized to $\tilde{o}$ and $\tilde{q}$, respectively. In query-aware search [26], we regard $h_a(\tilde{q})$ as the origin, i.e., $h_a(\tilde{q}) = 0$. Thus, for $\tilde{o}$ with distance $s = \|\tilde{o} - \tilde{q}\| = \sqrt{2(1 - \cos\theta)}$, we have

$$h_a(\tilde{o}) \sim \mathcal{N}(0, s^2). \tag{3.5}$$

Once we establish an (*anchor*) window with width $2w$ centered at the origin on the line of direction vector a, the probability that $h_a(\tilde{o})$ falls inside the anchor window is given by

$$P(h_a(\tilde{o}) \in [-w, w]) = f(w/s) = \frac{1}{\sqrt{2\pi}} \int_{-w/s}^{w/s} e^{-x^2/2} dx. \tag{3.6}$$

Let us rewrite this probability by $p(s, w)$ as

$$p(s, w) = f(w/s). \tag{3.7}$$

According to Equation (3.7), $p(s, w)$ is decreasing in s and increasing in w. For a set of hash functions $\mathcal{H} = \{h_{a_1}, h_{a_2}, \ldots, h_{a_m}\}$ with $\{a_j\}_{j=1}^m$ randomly generated, we denote the number of collisions (of o) by $\#\text{Col}(\tilde{o}, \tilde{q})$ (if $\tilde{o}$ falls inside window $[-w, w]$, we say that $\tilde{o}$ *collides* with $\tilde{q}$ in this window).

3.4.2.3 Fundamental relationships

Equation (3.6) indicates that we must control the window size $2w$ appropriately, depending on the distance s to ensure the validity of the collision testing. More concretely, the following question arises: what size of w is necessary to find a specific object o, with distance $s = \|\tilde{o} - \tilde{q}\|$, by the collision test with m hash functions? From (3.6) and Hoeffding's inequality on the sum of independent bounded random variables, for o with distance $s = \|\tilde{o} - \tilde{q}\|$ and w, we have

$$P(\#\mathrm{Col}(\tilde{o}, \tilde{q}) \geq m/2) \geq 1 - e^{-2m(p(s,w)-1/2)^2}. \tag{3.8}$$

According to Equation (3.8), for a given number of projections m and error rate δ ($0 < \delta < 1$), for the probability that o passes the collision test above at least $1-\delta$, we require

$$p(s, w) = f(w/s) \geq 1/2 + \sqrt{\frac{1}{2m} \log \frac{1}{\delta}}. \tag{3.9}$$

Let $p_0 = 1/2 + \sqrt{\frac{1}{2m} \log \frac{1}{\delta}}$ be the required rate of collision in a single projection. Hereinafter, we assume that m is set such that $p_0 < 1$ for a given δ. Therefore, Equation (3.9) means that w must satisfy

$$w/s \geq f^{-1}(p_0). \tag{3.10}$$

because f is bijective and monotonically increasing in w/s.

With notation $I(o) = \|o\| \cos\theta$, the distance s between $\tilde{o}$ and $\tilde{q}$ can be written as

$$s = \|\tilde{o} - \tilde{q}\| = \sqrt{2(1 - \cos\theta)} = \sqrt{2\,(1 - I(o)/\|o\|)}. \tag{3.11}$$

We define a function g as

$$g\left(\frac{I(o)}{\|o\|}; p_0\right) = \sqrt{2\left(1 - \frac{I(o)}{\|o\|}\right)} f^{-1}(p_0) = \sqrt{2 - 2\cos\theta} f^{-1}(p_0). \tag{3.12}$$

Therefore, condition (3.10) becomes

$$w \geq s f^{-1}(p_0) = g(I(o)/\|o\|; p_0). \tag{3.13}$$

That is, $g(I(o)/\|o\|; p_0)$ shows the necessary size of the window to find o with the cosine angle $\cos\theta = I(o)/\|o\|$ to q with the probability of at least $1 - \delta$. It is noteworthy that

$g(I/r; p)$ is bijective (in the range $0 \leq \theta \leq \pi$) in I/r, decreasing in I, and increasing in r. Furthermore, $g(I/r; p)$ is increasing in the parameter p. Hence, we have

$$g(I(o)/\left\|o\right\|; p_0) \leq g(I(o)/u_i; p_0), \quad o \in S_i. \tag{3.14}$$

Our goal is to determine an appropriate value of w for finding o^*. To achieve this, from (3.13), it suffices to determine the value of w such that

$$w \geq g(I(o^*)/\|o^*\|; p_0). \tag{3.15}$$

However, neither $I(o^*)$ nor $\|o^*\|$ is known in advance; hence, we cannot directly determine w by (3.15). We will describe a strategy to address this in the following section.

3.4.2.4 Search process

First, let us consider a simpler problem: if we know the subset S_{i^*} in which o^* lies, then how do we determine the value of w_{i^*} in S_{i^*}? This problem can be solved efficiently by the dynamic counting technique [20] as follows. At the beginning of the search, we set w_{i^*} to 0. As the search proceeds, we increase the value of w_{i^*} stepwise by Δw_{i^*}. If some object o passes the collision test ($\#\mathrm{Col}(\tilde{o}, \tilde{q}) \geq m/2$), we view o as a candidate for o^* and compute $I(o)$ (It is noteworthy that $\#\mathrm{Col}(\tilde{o}, \tilde{q})$ is nondecreasing in w_{i^*}). During this process, we update $\hat{o}$, the current most promising candidate of o^*, and the corresponding inner product $\hat{I} = I(\hat{o})$ when an object o with $I(o) > \hat{I}$ is found. Because $I(o^*) \geq \hat{I}$ and $\|o^*\| \leq u_{i^*}$, we have

$$g(\hat{I}/u_{i^*}; p_0) \geq g(I(o^*)/\left\|o^*\right\|; p_0). \tag{3.16}$$

Thus, for (3.15) to hold, it suffices to set w_{i^*} to $g(\hat{I}/u_{i^*}; p_0)$. The search step is illustrated in Figure 3.3.

Because the value of i^* is unknown, we must verify every i as follows: we set w_i ($1 \leq i \leq n$) to $T_i = g(\hat{I}/u_i, p_0)$, which results in w_{i^*} to T_{i^*}=$g(\hat{I}/u_{i^*}; p_0)$. If $o \in S_i$ satisfies $I(o) > \hat{I}$, then the normalized object $\tilde{o}$ passes the collision test with window size $2T_i$ and probability of at least $1-\delta$. As a special case, $I(o^*)$ is found in S_{i^*}. However, it is costly to use T_i directly because multiple objects pass the collision test with T_i. For a more efficient search process, we adopted a multi-round search strategy (Figure 3.2).

That is, in the rth round, we stopped the examination of S_i when w_i reached $t_i^r (< T_i)$; subsequently, we proceeded to S_{i+1}. Therefore, the problem became, "how does one set the values of t_i^r"? We considered an increasing sequence $\tau(1) < \tau(2) < \cdots < \tau(R) = 1$, where $\tau(r) = r/R$. Subsequently, based on the current $\hat{I}$, which was appropriately initialized, we determined an upper bound t_i^r of w_i at round r as follows. First, we estimated the expected inner product $I_b(r)$ in S_1 at round r such that

$$I_b(r)/u_1 = g^{-1}(g(\hat{I}/u_1; \tau(r)); 1/2). \tag{3.17}$$

Here, for a fixed value of x, because $g^{-1}(g(\alpha; \tau), 1/2)$ is decreasing in τ, we have $1 > I_b(1) > \cdots > I_b(R) = -\infty$ (Figure 3.4).

Subsequently, we determine the value of t_i^r $(1 \leq i \leq n)$ by

$$t_i^r = g(I_b(r)/u_i; 1/2). \tag{3.18}$$

Therefore, we share the same criterion, that is, $I_b(r)$ in all subsets $S_1, \cdots, S_n$. This t_i^r guarantees that an object $o \in S_i$ with $I(o) \geq I_b(r)$ has the expected collision number of at least $m/2$ if $\tau(r) > 1/2$. This can be confirmed from $t_i^r = g(I_b(r)/u_i; 1/2) \geq g(\hat{I}/u_i; 1/2) \geq g(I(o)/\|o\|; 1/2)$ for $\tau(r) > 1/2$ (the first inequality is derived from Equations (3.17)). This multi-round search is expected to accelerate the termination in an earlier round. Furthermore, because we set $\tau(R)$ to 1 and derived $I_b(R) = -\infty$ and $t_i^R = +\infty$ $(1 \leq i \leq n)$, the termination condition $w_i \geq g(\hat{I}/u_i; p_0) = T_i$ is satisfied before reaching the Rth round (Figure 3.3).

Next, we describe the complete search process (Figure 3.2). We divide the search process into R rounds and introduce a round bound t_i^r for w_i $(i = 1, \cdots, n)$ at the rth round by Equations (3.17) and (3.18). At the rth round $(r = 1, 2, \cdots, R)$ in S_i $(i = 1, \cdots, n)$, we verify every object that passes the collision test in $[-w_i, w_i]$ (a candidate) and then increase the value of w_i gradually by Δw until w_i reaches t_i^r. When w_i exceeds t_i^r, we proceed to S_{i+1}. After S_n is examined in this round, we return to S_1 and restart the search at the $(r+1)$th round.

In the middle of the search process, we can terminate the examination of S_i if one of the following three termination conditions is satisfied.

TC1. We can skip the examination of $S_i, S_{i+1}, \ldots, S_n$, if $u_i < \hat{I}$. This is because any $o \in \cup_{j \geq i} S_j$ satisfies $\|o\| \leq u_i$; thus, $I(o) = \|o\| \cos\theta \leq u_i$.

TC2. If $w_i \geq g(\hat{I}/u_i; p_0) = T_i$, we can terminate the search in S_i because (3.15) is satisfied (see Figure 3.3). This is a special case of the following condition, TC3.

TC3. For any c $(0 < c \leq 1)$, we can terminate the search in S_i if $w_i \geq g(\hat{I}/(cu_i); p_0)$. The reason will be explained in Theorem 5 later.

3.4.2.5 Performance analysis

The details of the query phase are shown in Algorithm 5. Let us analyze its time complexity. In the worst case, we must scan the entire hash tables and compute the exact inner products between q and all objects in $\mathcal{D}$, which means that the time complexity in the worst case is $O(Nd)$, where N is the size of $\mathcal{D}$ and d is the data dimension. This is intuitive as AdaLSH can support an exact MIPS. However, AdaLSH performs excellently for real datasets because the worst case does not occur in practice usually.

We can guarantee the performance of AdaLSH as follows:

Theorem 5. *For a given error rate δ $(0 < \delta < 1)$, AdaLSH solves the c-AMIP problem AMIP($\mathcal{D}, q, c$) with the probability of at least $1 - \delta$, where $0 < c \leq 1$.*

Proof. First, the main loop steps (7–21) verifies every i in the order of $i = 1, 2, \ldots, n$. For each subset S_i, the examination of S_i can be terminated when either TC1 or condition TC3 (TC2 is a special case) is satisfied (it is noteworthy that t_i^R is $+\infty$ in the Rth round, which ensures for each S_i that either of these two conditions is satisfied). However, only when the examinations of all subsets have been terminated can we terminate the query phase. Thus, it suffices to consider S_{i^*} only. Condition TC1 cannot be applied to S_{i^*} because $u_{i^*} \geq I(o^*) \geq \hat{I}$. When condition TC3 is applied to S_{i^*}, either $w_{i^*} \geq g(I(o^*)/\left\|o^*\right\|; p_0)$ or $w_{i^*} \leq g(I(o^*)/\left\|o^*\right\|; p_0)$ holds. According to (3.15), the former case implies that we can find o^* with the probability of at least $1 - \delta$. In the latter case, we have $g(\hat{I}/(cu_{i^*}); p_0) \leq w_{i^*} \leq g(I(o^*)/\left\|o^*\right\|; p_0) \leq g(I(o^*/u_{i^*}; p_0))$. This means that $\hat{I} = I(o) \geq cI(o^*)$, implying that some object o has already been found such that $I(o) \geq cI(o^*)$. Thus, regardless of the case, we can find a c-AMIP object with the probability of at least $1 - \delta$. □

Although we have not considered the case $\hat{I} < 0$, it suffices to replace the termination condition in Step #19 with $w_i \geq g(c\hat{I}/\ell_i; p_0)$ in the case thereof. Under this modification, Theorem 5 still holds. For finding k $(k > 1)$ c-AMIP objects, we must maintain k MIP objects among the candidates and replace $\hat{I}$ in Algorithm 5 with

$\hat{I}_k = I(\hat{o}_k)$, where $\hat{o}_k$ is the kth MIP object among the candidates. By applying our discussion on o^* to o_i^*, where o_i^* is the true ith MIP object, Theorem 5 can be easily extended for the c-k-AMIP problem. In addition, we have the following results regarding recalls:

Corollary 2. *For a given error rate δ ($0 < \delta < 1$), AdaLSH can find k MIP objects with the expectation of the recall rate of at least $1 - \delta$ when $c = 1$.*

3.4.3 Comparison with other LSH methods

In this subsection, we compare AdaLSH with other LSH methods with probability guarantees from the theoretical perspective. For this goal, we selected H2-ALSH as the representative because it has a better theoretical guarantee than other methods [27].

(1) **The guaranteed success probability.** As shown in Algorithm 5, AdaLSH supports arbitrary error rate. To guarantee this error, it is sufficient to ensure that p_0 lies in the interval (0,1) such that the computation of $g(\hat{I}/(cu_i); p_0)$ is practical. This is feasible because the value of p_0 depends on m, that is controllable by choosing an appropriate value of p_0. In contrast, the error rate of H2-ALSH cannot be less than $1/2 - 1/e$. This is because (1) the probability guarantee of H2-ALSH was obtained in the worst case, and in the searching phase, the full information was not exploited, and (2) many loose inequalities were used in the derivation of its theoretical guarantee.

(2) **The tuning of approximation ratio c.** As shown in Algorithm 5, AdaLSH terminates when the inequality $w_i \geq g(\hat{I}/(cu_i); p_0)$ holds. Since function $g(x, y)$ is decreasing in x given y, the value of $g(\hat{I}/(cu_i); p_0)$ can be controlled by adjusting c. In other words, AdaLSH can terminate earlier by selecting a smaller c. H2-ALSH also supports any value of c in $(0, 1)$, but the algorithm cannot gain efficiency even for a small value of c due to its theoretical limitations. Relevant experiments are shown in Sec. 5.

3.5 Experimental evaluation

AdaLSH was implemented in C++. All the experiments were performed on a PC with Intel(R), 3.40 GHz i7-4770 eight-core processor with 8 GB RAM, in Ubuntu 16.04.

3.5.1 Experimental setup

3.5.1.1 Datasets and queries

We selected ten real datasets for the experiments (Table 3.3). Let $\lambda = \min \|o\| \,/\, \max \|o\|$ ($o \in D$) be the ratio of the minimum norm to the maximum norm over all data in the dataset. If for any dataset on which $\lambda \geq b$ holds ($b = r_i/r_{i-1}$), that is, all objects fall in the same ring on this dataset, we call it a *homocentric dataset*, otherwise, we call it a *non-homocentric dataset*. Among these ten real datasets, **Deep** and **Ukbench** are homocentric, and the others are non-homocentric. Note that although **ImageNet** is non-homocentric in the definition, it is homocentric in essence because most of the objects fall in the same ring. Thus, we consider **ImageNet** as a homocentric dataset. In addition, for each dataset, we selected 200 queries randomly from its corresponding query set and generated the ground truth set in advance by a linear scan. In this section, all results are the average results over 200 queries.

- **Cifar**[1] is a labeled subset of TinyImage dataset, with each image represented by a 512-d GIST feature vector.
- **Sun**[2] comprises GIST features of images.
- **Enron**[3] originates from a collection of emails and contains feature vectors of bi-grams.
- **Trevi**[4] comprises 0.4 million 1024 bitmap(.bmp) images, each containing a 16 $\times$ 16 array of image patches.
- **Nusw** [5] includes approximately 2.7 million web images, each as a 500-dimensional bag-of-words vector.
- **Msong**[6] is a collection of audio features and metadata for 1 million contemporary popular music tracks.
- **Gist**[7] is an image dataset comprising approximately 1 million data points.
- **Ukbench**[8] contains approximately 1 million 128-d features of images.

[1] http://www.cs.toronto.edu/ kriz/cifar.html
[2] http://groups.csail.mit.edu/vision/SUN/
[3] http://www.cs.cmu.edu/ enron/
[4] http://phototour.cs.washington.edu/patches/default.htm
[5] https://lms.comp.nus.edu.sg/research/NUS-WIDE.htm
[6] http://www.ifs.tuwien.ac.at/mir/msd/download.html
[7] http://corpus-texmex.irisa.fr/
[8] http://vis.uky.edu/ stewe/ukbench/

- **Deep**[1] contains deep neural codes of natural images obtained from the activations of a convolutional neural network.
- **ImageNet** [2] is introduced and employed by "The ImageNet Large Scale Visual Recognition Challenge", which contains approximately 2.4 million data points with 150 dimensions of dense SIFT features.

3.5.1.2 Performance metrics

We selected the following performance metrics in our experiments. The overall ratio and recall were used because they were related to the probability guarantees of AdaLSH.

- **Approximation ratio (c)**. Each of answered k approximate MIP objects has to be close to the corresponding true MIP object in inner product with at least ratio c ($0 < c \leq 1$), where c is given by the user.
- **Running time**. It indicates the time consumed in the query phase. The faster is, the better as long as approximation condition is satisfied.
- **Overall ratio**. It indicates the average of approximation ratios of answered k MIP objects. For c-k-MIPS, the overall ratio is defined as $\frac{1}{k}\sum_{i=1}^{k} \langle \hat{o}_i, q\rangle \,/\, \langle o_i^*, q\rangle$. Note that this values is not directly related to the approximation ratio c. For example, a high value of overall ratio, say 0.9, for $c = 0.5$, can be worse than 0.6, if a single MIP object is approximated only by 0.4 for $c = 0.5$.

In addition, we use the following metrics for two-class (positive and negative) problems.

- **Precision**. It indicates the ratio of correct answers to all answers.
- **Recall rate**. It indicates the ratio of correct answers to positive data.
- **PR curve**. It shows the relationship between precision and recall. The higher the curve is, the better the corresponding method is.

3.5.1.3 Parameter setting of AdaLSH

The parameters to be determined are b and m, where m is the number of generated hash functions and b denotes the ratio of the radii of two neighbor rings, i.e., $b = r_i/r_{i-1}$.

[1] https://yadi.sk/d/I_yaFVqchJmoc
[2] https://cloudcv.org/objdetect/

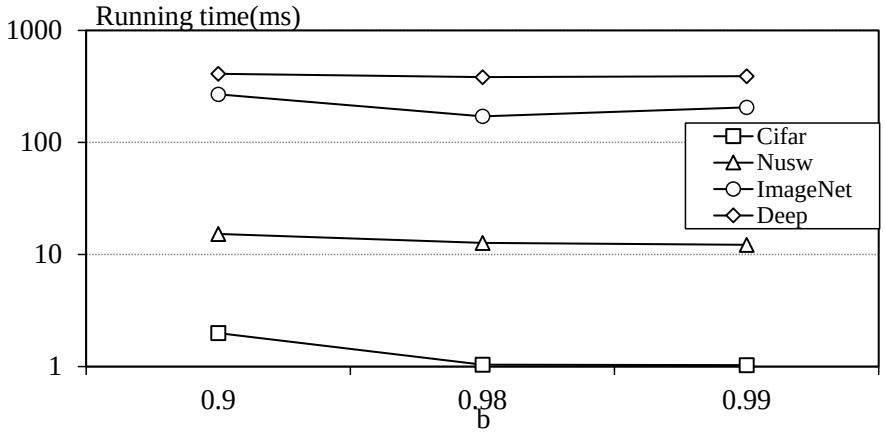


Figure 3.5: Performances of AdaLSH under different b ($c = 1.0$, $k = 100$)

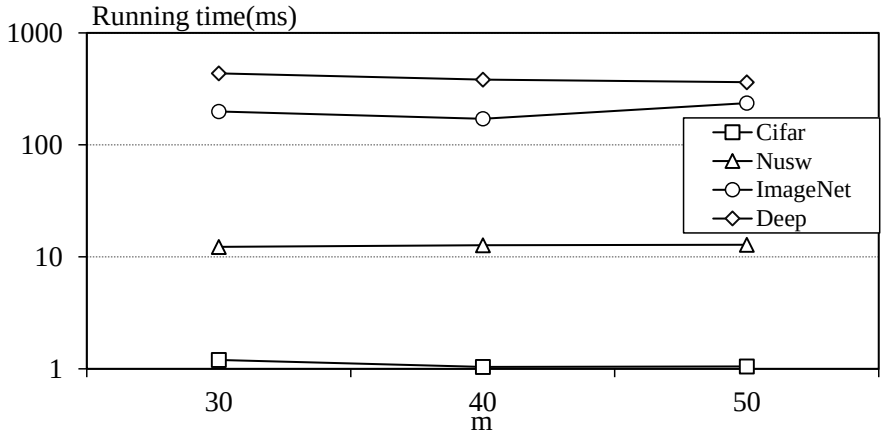


Figure 3.6: Performances of AdaLSH under different m ($c = 1.0$, $k = 100$)

Figures 3.5 and 3.6 show the speeds of AdaLSH under different combinations of b and m (the recalls were omitted because they were all close to 100%). Here, we only show the results on **Cifar**, **Nusw**, **Deep**, and **ImageNet**. Similar trends were observed on other datasets. From the results, it is clear that (1) the performances of AdaLSH are not highly sensitive to the change in these two parameters, and (2) $b = 0.98$, $m = 40$ are suitable settings because AdaLSH runs the fastest in this case. Therefore, hereinafter, we use $b = 0.98$, $m = 40$ as our default setting unless stated otherwise.

3.5.2 Efficiency of AdaLSH

In this section, we will describe the efficiency of AdaLSH from the following two aspects: (1) the effect of the multi-round search strategy, and (2) the tuning capability based

on the parameters.

3.5.2.1 Efficiency of multi-round strategy

We first used an artificial dataset to confirm the advantage of the multi-round search strategy (a small portion of each S_i was examined in each round) used in AdaLSH against its counterpart, i.e., the single-round search (adopted by H2-ALSH). We randomly generated a query q and $2 \cdot 10^5$ objects of D such that their angles to q were larger than $7\pi/18$. By adjusting their norms, we halved D into S_1 and S_2 such that objects in S_1 have larger norms than those in S_2. Furthermore, we added object o^* such that it was in S_2 (closer to the origin), and its angle to q was one of three candidate angles $\{\ \pi/18,\ \pi/6,\ 5\pi/18\}$. Thus, we generated a situation where the angle of o^* to q was smaller than those of the other $o \in D$, but the norm $\|o^*\|$ was relatively small. For each candidate angle, we generated the MIP object o^* 50 times and conducted an exact MIPS by AdaLSH ($c = 1.0$, $\delta = 0.1$) on the same D and q. The results are shown in Table 3.4. First, we observe that all MIP objects were found at 100% accuracy, even for the required value of 90% ($= 1 - \delta$). In Table 3.4, the column of S_1 shows the searching time of S_1 only (it is noteworthy that in the single-round search, we did not proceed to S_2 until the entire S_1 was examined). The fact that the total running time (28.17 ms or 12.58 ms) of the $\pi/6$ or $\pi/18$ case is less than the searching time of S_1 (= 44.49 ms) demonstrates that we can find MIP objects with small norms earlier by the multi-round search strategy. Hence, the search process can be accelerated efficiently.

3.5.2.2 Performance of AdaLSH under different parameters

For several combinations of k, c, and δ, we present the speeds of AdaLSH in Figure 3.7 and the recall rates in Figure 3.8. Here, we only present the results on **Nusw** because similar trends were observed on other datasets. As shown in Figure 3.7, (1) as k increases, AdaLSH requires more query time. This is a natural expense for finding more MIP objects; (2) the smaller the c and the larger the δ, the faster is AdaLSH. Particularly, it is clear that the decrease in c is more effective than the increase in δ.

Figure 3.8 shows that a higher recall rate can be obtained by increasing the value of c or by decreasing the value of δ. Although both c and δ can affect the query accuracy, the performance of AdaLSH is more sensitive to the change in c. Combined with the

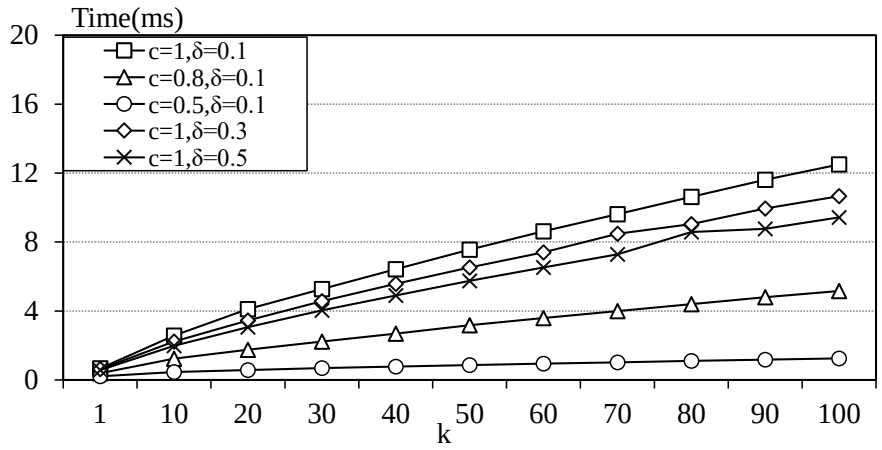


Figure 3.7: Speeds of AdaLSH under different parameters on Nusw. The naive search consumed approximately 107 ms

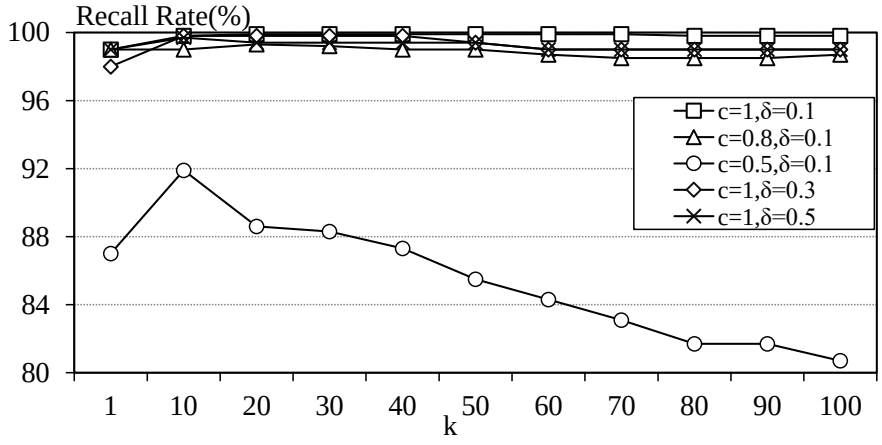


Figure 3.8: Recall rates of AdaLSH under different parameters on Nusw

observations above, we conclude that the tuning of c is important for achieving different tradeoffs between cost and quality.

3.5.3 The comparison study

In this subsection, we compare AdaLSH with other state-of-the-art MIPS methods. Since the performance metrics mentioned in Sec. 3.5.1.3 do not apply to each method due to their different proposals, we divide them into three classes and compared our method with them separately. In Sec. 3.5.3.1, we compare AdaLSH with L2-ALSH [48], Simple-ALSH [43], Sign-ALSH [49], and Xbox [6]. These four compared methods are learning-free, as mentioned previously; however, they cannot support the arbitrary approximation ratio c. Thus, we conduct the experiments under $c = 0.5$, as suggested by their authors. In Sec. 3.5.3.2, we compare AdaLSH with H2-ALSH, a state-of-the-art LSH method that can support any c in (0,1).

The internal parameters of the benchmark methods were all set suitably, as suggested by their authors. More specifically, we selected $c = 0.5$ as the standard setting of L2-ALSH, Simple-ALSH, and Sign-ALSH, and we selected 64 bits as the binary code length for AIBC. For AdaLSH, b and m were set to 0.98 and 40, respectively, as shown in Sec. 3.5.1.3, and the error rate δ was fixed to 0.1 in all experiments.

3.5.3.1 AdaLSH vs. other LSH-based methods

The experimental results are listed in Table 3.5 and Table 3.6. The summary of the results is as follows:

(1) On all non-homocentric datasets except **Trevi**, AdaLSH runs the fastest and achieves the highest recall rates. This shows that by selecting a suitable value of c, say 0.5, AdaLSH can perform more efficiently than other LSH-based methods on non-homocentric datasets.

(2) On homocentric datasets **Ukbench** and **Deep**, Sign-ALSH and SimpleLSH outperformed AdaLSH because the homocentric hypersphere structure loses its effectiveness on these two datasets. However, as will be shown later, even on these two datasets, AdaLSH can achieve high recalls by selecting a larger c, say 0.99.

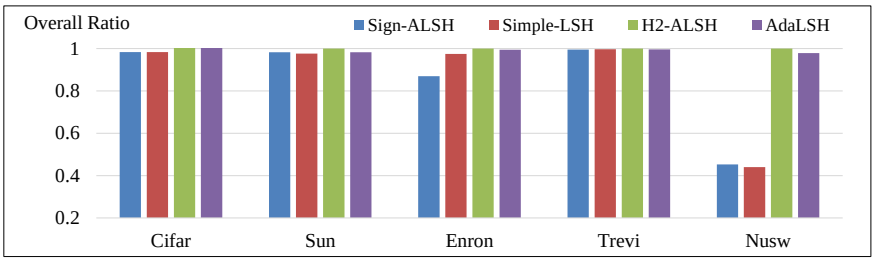


Figure 3.9: Comparison of overall ratios on Cifar, Sun, Enron, Trevi, and Nusw (k = 100, c = 0.5)

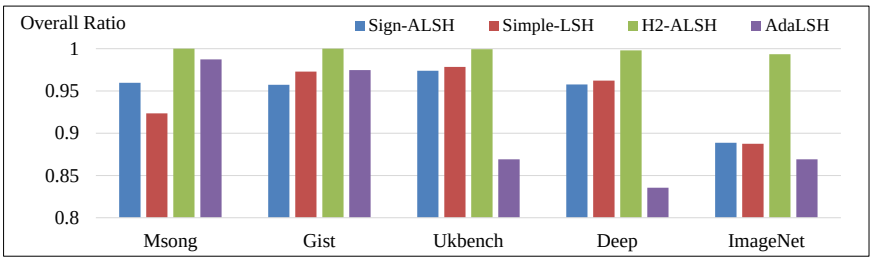


Figure 3.10: Comparison of overall ratios on Msong, Gist, Ukbench, Deep, and ImageNet (k = 100, c = 0.5)

3.5.3.2 AdaLSH vs. H2-ALSH

Next, we replaced L2-ALSH with H2-ALSH, and compared their overall ratios, recall and time. We considered two cases: a loose requirement of $c = 0.5$ and a tight requirement of $c = 0.99$ (H2-ALSH cannot support $c = 1$). Here, $c = 0.5$ is the recommended setting of H2-ALSH. The results are shown in Figure 3.9, Figure 3.10, and Table 3.7. Note that it is important that the values of overall ratio are greater than a given approximate rate c regardless of large and small of the values. In addition, we have the following.

(1) Under $c = 0.5$, the overall ratios of H2-ALSH and AdaLSH are higher than the required value ($c = 0.5$), which is consistent with the theoretical guarantees. Meanwhile, AdaLSH requires a significantly shorter running time to achieve the goal of 0.5-ANNS than H2-ALSH, especially on nonhomocentric datasets. This is because the termination condition of AdaLSH is adjusted dynamically based on the current search results; therefore, AdaLSH terminates much earlier than H2-ALSH when the search goal has been achieved.

(2) Under $c = 0.99$, both H2-ALSH and AdaLSH can achieve high recalls (over 99%)

on non-homocentric datasets; meanwhile, on homocentric datasets, AdaLSH achieves a higher recall (99%) than H2-ALSH (91% and 96%). As for speed, AdaLSH is faster than H2-ALSH on non-homocentric datasets, but slower on homocentric datasets. This is due to the multi-round search strategy of AdaLSH.

(3) It is clear that the recall rate and speed of H2-ALSH do not change largely even if the value of c is increased from 0.5 to 0.99. This shows the inadaptability of H2-ALSH to the parameter change. In contrast, AdaLSH adopts the change of the value of c and succeed in gaining efficiency. Table 3.7 shows that, when $c = 0.5$, AdaLSH only requires 10%-25% of running time of H2-ALSH on most non-homocentric datasets.

Algorithm 5: Query phase of AdaLSH

Input: q: query;
c: approximation ratio ($0 < c \leq 1$);
δ: error rate($0 < \delta < 1$);
m: the number of generated hash functions;
R: maximum round times;
$\{(S_i, u_i, \ell_i)\}_{i=1}^{n}$: divided subsets of objects;
$\mathcal{H} = \{h_{a_j}\}_{j=1}^{m}$: hash functions;
Output: $\hat{o}$: a c-approximate MIP object for q;

$\mathcal{S} \leftarrow \{1, 2, \cdots, n\}$;
$\hat{I} \leftarrow \langle o', \tilde{q} \rangle$,where $o' = \arg\max_{o} \|o\|$;
$w_i \leftarrow 0$ $(i = 1, 2, \ldots, n)$; $r \leftarrow 1$;
$p_0 \leftarrow 1/2 + \sqrt{(1/(2m)) \log(1/\delta)}$;
while $\mathcal{S} \neq \emptyset$ **and** $r \leq R$ **do**
  $I_b \leftarrow u_1 g^{-1}(g(\hat{I}/u_1; \tau(r)); 1/2)$; % *see (13)*
  for $i \in \mathcal{S}$ *in ascending order* **do**
    if $u_i < \hat{I}$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} \backslash \{i\}$;
      break
    $t_i \leftarrow g(I_b/u_i; 1/2)$; % *see (14)*
    while $w_i \leq t_i$ **do**
      $w_i \leftarrow w_i + \Delta w_i$; % *extend windows*
      for $o \in S_i$ **do**
        if $\#Col(\tilde{o}, \tilde{q}) \geq m/2$ **then**
          $I \leftarrow \langle o, \tilde{q} \rangle$;
          if $I > \hat{I}$ **then**
            update $\hat{o}$ and $\hat{I}$ with o and I;
      if $w_i \geq g(\hat{I}/(cu_i); p_0)$ **then**
        $\mathcal{S} \leftarrow \mathcal{S} \backslash \{i\}$;
        break
  $r \leftarrow r + 1$
return $\hat{o}$

Table 3.3: Real datasets

Dataset	Dimension	Size($\times 10^3$)	Type
Cifar	512	50	Image
Sun	512	79	Image
Enron	1369	95	Text
Trevi	4096	100	Image
Nusw	500	269	Image
Msong	420	922	Audio
Gist	960	1000	Image
Ukbench	128	1000	Image
Deep	256	1000	Image
ImageNet	150	2340	Image

Table 3.4: Results on the artificial data (ms). The single-round search consumed 44.49 ms on S_1 (the 1st column), while the multi-round search examined S_1 (and S_2) piecewise in the ascending order of the angle (2nd–4th columns).

Angle	S_1	$5\pi/18$	$\pi/6$	$\pi/18$
Running time	44.49	61.93	**28.17**	**12.58**
Recall rate	–	100	100	100

Table 3.5: Recall rate(%) (k=100, c=0.5)

Dataset	Xbox	L2-ALSH	Simple-LSH	Sign-ALSH	AdaLSH
Cifar	36.46	41.13	67.98	68.93	**79.16**
Sun	25.58	26.34	59.90	64.56	**64.85**
Enron	1.11	11.69	83.71	78.76	**97.32**
Trevi	43.02	62.45	**83.85**	79.25	79.28
ImageNet	1.46	8.08	33.07	31.84	**35.59**
Gist	25.92	32.42	68.75	61.71	**69.34**
Ukbench	52.55	38.41	**49.73**	43.91	15.17
Nusw	13.92	9.57	15.95	20.91	**80.77**
Msong	53.19	7.11	70.87	80.42	**89.71**
Deep	33.95	25.68	**42.20**	38.54	8.07

Table 3.6: Running time(ms) (k=100, c=0.5)

Dataset	Xbox	L2-ALSH	Simple-LSH	Sign-ALSH	AdaLSH
Cifar	6.01	4.81	4.12	4.11	**0.08**
Sun	11.31	9.68	6.34	6.27	**0.09**
Enron	17.04	21.44	7.91	7.93	**0.19**
Trevi	15.80	8.09	9.63	9.62	**0.49**
ImageNet	1250.28	1232.72	176.42	176.11	**35.95**
Gist	310.10	331.57	85.38	86.78	**0.97**
Ukbench	173.11	181.12	82.81	**82.64**	88.97
Nusw	62.55	58.73	20.61	20.58	**1.39**
Msong	264.31	221.94	75.19	74.77	**0.51**
Deep	164.63	175.51	75.62	**75.57**	95.71

Table 3.7: AdaLSH vs. H2-ALSH (k=100)

Dataset	$c = 0.5$				$c = 0.99$			
	Recall(%)		Speed(ms)		Recall(%)		Speed(ms)	
	H2ALSH	AdaLSH	H2ALSH	AdaLSH	H2ALSH	AdaLSH	H2ALSH	AdaLSH
Cifar	99	79	1.04	0.08	100	100	2.11	1.05
Sun	99	65	0.94	0.09	99	100	1.80	1.06
Enron	100	97	0.23	0.19	100	100	0.25	0.23
Trevi	100	79	2.46	0.49	100	100	2.70	1.83
ImageNet	86	36	51.35	35.95	98	99	151.32	171.31
Gist	99	69	26.83	0.97	100	100	39.26	20.24
Ukbench	96	11	95.75	88.97	96	99	100.01	241.38
Nusw	99	81	8.42	1.39	100	99	23.49	12.26
Msong	100	90	2.19	0.51	100	100	3.53	2.30
Deep	91	8	100.11	95.71	91	99	102.48	381.33

Chapter 4

Conclusion and Future Work

In this paper, we comprehensively review the applications of LSH in two spaces: the Euclidean space with ℓ_2 metric (the first part) and the inner product space (the second part). In the first part, we proposed two novel LSH based methods VHP and R2LSH. Compared with existing methods, VHP still works on a group of one-dimensional hash functions but utilize more projection information, that is, partial distances, while R2LSH works in multiple 2-dimensional subspaces, which could distinguish true nearest neighbors much accurately than classical LSH methods based on one-dimensional projected vectors. Both proposed methods could work with an arbitrarily small approximation ratio $c \geq 1$ and is guaranteed to identify c-k-ANN with the specified success probability. Extensive experiments show that, compared with state-of-the-art LSH methods SRS and QALSH, VHP and R2LSH save significant I/O costs and running times under identical answer quality.

In the second part, we herein proposed a novel search method named AdaLSH that supported both exact and approximate MIPS with a strict probability guarantee. Compared with state-of-the-art LSH methods, AdaLSH demonstrated a tighter probability guarantee in the general case, which filled the gap between the real performance and the theoretical guarantee, thereby allowing users to predict the query results more precisely. In addition, from the experimental results, it was clear that AdaLSH performed excellently on non-homocentric datasets owing to the efficient search strategy. That is, compared with other state-of-the-arts LSH methods, our method required a shorter running time but achieved a higher (or close) recall. Therefore, we recommend users to use our method on datasets of such characteristics. Meanwhile, for homocentric datasets, although the performance of AdaLSH degraded significantly, AdaLSH could

still achieve high recalls by selecting large approximation ratios, owing to the theoretical guarantee. Thus, users are recommended to use our method on homocentric datasets if top priority is given to precision.

For future studies, I think it is promising to apply some core ideas of LSH to ANN search methods of other types. For example, in recent years, graph-based methods have shown their superiority over many other search methods. However, the research on how to select the connected edges for each node, which represents a data point, is sill in progress. If we can select several data-dependent hash functions, we can partition the neighborhood of each node into many cells. Then we select connected neighbors such that they are contained in different cells. In this way, we can ensure that the distances among connected neighbors are large enough to improve the search efficiency.

Bibliography

[1] A. Andoni and P. Indyk. E2lsh manual. In *http://web.mit.edu/andoni/www/LSH*.

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

[3] A. Andoni, P. Indyk, T. Laarhoven, I. P. Razenshteyn, and L. Schmidt. Practical and optimal LSH for angular distance. In *NIPS*, pages 1225–1233, 2015.

[4] W. G. Aref, A. C. Catlin, J. Fan, A. K. Elmagarmid, M. A. Hammad, I. F. Ilyas, M. S. Marzouk, and X. Zhu. A video database management system for advancing video database research. In *Multimedia Information Systems*, pages 8–17, 2002.

[5] A. Arora, S. Sinha, P. Kumar, and A. Bhattacharya. Hd-index: Pushing the scalability-accuracy boundary for approximate knn search in high-dimensional spaces. *PVLDB*, 11(8):906–919, 2018.

[6] Y. Bachrach, Y. Finkelstein, R. Gilad-Bachrach, L. Katzir, N. Koenigstein, N. Nice, and U. Paquet. Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces. In *Eighth ACM Conference on Recommender Systems*, pages 257–264, Foster City, Silicon Valley, CA, USA, October 2014. ACM.

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

[8] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS*, pages 585–591, 2001.

[9] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1930–1941, 2013.

[10] J. L. Bentley. K-d trees for semidynamic point sets. In *SoCG*, pages 187–197, 1990.

[11] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. *Computer Networks*, 29(8-13):1157–1166, 1997.

[12] A. Chakrabarti and O. Regev. An optimal randomised cell probe lower bound for approximate nearest neighbour searching. In *FOCS*, pages 473–482, 2004.

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

[14] P. Ciaccia and M. Patella. PAC nearest neighbor queries: Approximate and controlled search in high-dimensional and metric spaces. In *ICDE*, pages 244–255, 2000.

[15] A. C. Cohen. Truncated and censored samples : theory and applications. *CRC Press*, 1991.

[16] R. R. Curtin, P. Ram, and A. G. Gray. Fast exact max-kernel search. *Statistical Analysis and Data Mining*, 7(1):1–9, February–December 2014.

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

[18] T. Dean, M. Ruzon, M. Segal, J. Shlens, S. Vijayanarasimhan, and J. Yagnik. Fast, accurate detection of 100,000 object classes on a single machine. In *CVPR*, pages 1814–1821, 2013.

[19] K. Echihabi, K. Zoumpatianos, T. Palpanas, and H. Benbrahim. Return of the lernaean hydra: Experimental evaluation of data series approximate similarity search. *PVLDB*, 13(3):403–420, 2019.

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

[21] 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.

[22] J. Gao, H. V. Jagadish, B. C. Ooi, and S. Wang. Selective hashing: Closing the gap between radius search and k-nn search. In *SIGKDD*, pages 349–358, 2015.

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

[24] H.Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24:417–441, 1933.

[25] Q. Huang, J. Feng, Q. Fang, W. Ng, and W. Wang. Query-aware locality-sensitive hashing scheme for l_p norm. *VLDB J.*, 26(5):683–708, 2017.

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

[27] Q. Huang, G. Ma, J. Feng, Q. Fang, and A. K. H. Tung. Accurate and fast asymmetric locality-sensitive hashing scheme for maximum inner product search. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1561–1570, London, UK, August 2018. ACM.

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

[29] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(1):117–128, 2011.

[30] T. Joachims, T. Finley, and C.-N. J. Yu. Cutting-plane training of structural svms. *Machine Learning*, 77(1):27–59, October–December 2009.

[31] N. Katayama and S. Satoh. The sr-tree: An index structure for high-dimensional nearest neighbor queries. In *SIGMOD*, pages 369–380. ACM Press, 1997.

[32] Y. Ke, R. Sukthankar, and L. Huston. An efficient parts-based near-duplicate and sub-image retrieval system. In *ACM Multimedia*, pages 869–876, 2004.

[33] J. M. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. In *STOC*, pages 599–608, 1997.

[34] N. Koenigstein, P. Ram, and Y. Shavitt. Efficient retrieval of recommendations in a matrix factorization framework. In *CIKM*, pages 535–544, 2012.

[35] W. Li, Y. Zhang, Y. Sun, W. Wang, W. Zhang, and X. Lin. Approximate nearest neighbor search on high dimensional data - experiments, analyses, and improvement. *CoRR abs*, 2016.

[36] 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.

[37] K. Lu and M. Kudo. R2lsh: A nearest neighbor search scheme based on two-dimensional projected spaces. In *ICDE*, pages 1045–1056, 2020.

[38] K. Lu and M. Kudo. Adalsh: Adaptive lsh for solving c-approximate maximum inner product search problem. *IEICE Trans. Inf. Syst.*, 104-D(1):138–145, 2021.

[39] K. Lu, H. Wang, W. Wang, and M. Kudo. Vhp: Approximate nearest neighbor search via virtual hypersphere partitioning. *PVLDB*, 13(9):1443–1455, 2020.

[40] 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.

[41] Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *CoRR*, abs/1603.09320, 2016.

[42] M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(11):2227–2240, 2014.

[43] B. Neyshabur and N. Srebro. On symmetric and asymmetric lshs for inner product search. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 192–1934, Lille, France, July 2015.

[44] P. Ram and A. G. Gray. Maximum inner-product search using cone trees. In *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 931–939, Beijing, China, August 2012. ACM.

[45] S. Riedel, L. Yao, B. M. Marlin, and A. McCallum. Relation extraction with matrix factorization and universal schemas. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics*, pages 74–84, Atlanta, Georgia, USA, June 2013. The Association for Computational Linguistics.

[46] B. Scholkopf, A. J.Smola, and K.-R. Muller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998.

[47] F. Shen, Y. Yang, L. Liu, W. Liu, D. Tao, and H. T. Shen. Asymmetric binary coding for image search. *IEEE Trans. Multimedia*, 19(9):2022–2032, 2017.

[48] A. Shrivastava and P. Li. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *Annual Conference on Neural Information Processing Systems*, pages 2321–2329, Montreal, Quebec, Canada, December 2014.

[49] A. Shrivastava and P. Li. Improved asymmetric locality sensitive hashing (alsh) for maximum inner product search (mips). In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*, pages 812–821, Amsterdam, The Netherlands, July 2015. AUAI Press.

[50] R. Spring and A. Shrivastava. Scalable and sustainable deep learning via randomized hashing. In *KDD*, pages 445–454, 2017.

[51] 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.

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

[53] C. Teflioudi and R. Gemulla. Exact and approximate maximum inner product search with lemp. *ACM Trans. Database System*, 42(1):1–49, February–December 2017.

[54] J. Wang, T. Zhang, J. Song, N. Sebe, and H. T. Shen. A survey on learning to hash. *IEEE Trans. Pattern Anal. Mach. Intell.*, 40(4):769–790, 2018.

[55] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, pages 194–205. Morgan Kaufmann, 1998.

[56] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, pages 1753–1760, 2008.

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