VHP: Approximate Nearest Neighbor Search via Virtual Hypersphere Partitioning

Kejing Lu[†], Hongya Wang[‡][*], Wei Wang[§], Mineichi Kudo[†]
[†]Graduate School of Information Science and Technology, Hokkaido University, Japan
[‡]School of Computer Science and Technology, Donghua University, China
[§]School of Computer Science and Engineering, University of New South Wales, Australia
{lkejing,mine}@ist.hokudai.ac.jp, hywang@dhu.edu.cn, weiw@unsw.edu.au

ABSTRACT

Locality sensitive hashing (LSH) is a widely practiced c-approximate nearest neighbor(c-ANN) search algorithm in high dimensional spaces. The state-of-the-art LSH based algorithm searches an unbounded and irregular space to identify candidates, which jeopardizes the efficiency. To address this issue, we introduce the concept of *virtual hypersphere partitioning*. The core idea is to impose a virtual hypersphere, centered at the query, in the original feature space and only examine points inside the hypersphere. The search space of a hypersphere is isotropic and bounded, and thus more efficient than the existing one. In practice, we use multiple physical hyperspheres with different radii in corresponding projection subspaces to emulate the single virtual hypersphere. We also developed a principled method to compute the hypersphere radii for given success probability.

Based on virtual hypersphere partitioning, we propose a novel disk-based indexing and searching scheme VHP to answer c-ANN queries. In the indexing phase, VHP stores LSH projections with independent B^+-trees. To process a query, VHP keeps increasing the radii of physical hyperspheres coordinately, which in effect amounts to enlarging the virtual hypersphere, to accommodate more candidates until the success probability is met. Rigorous theoretical analysis shows that the proposed algorithm supports c-ANN search for arbitrarily small $c \geq 1$ with probability guarantee. Extensive experiments on a variety of datasets, including the billion-scale ones, demonstrate that VHP could achieve different tradeoffs between efficiency and accuracy, and achieves up to 2x speedup in running time over the state-of-the-art methods.

PVLDB Reference Format:
Kejing Lu, Hongya Wang, Wei Wang, Mineichi Kudo. VHP: Approximate Nearest Neighbor Search via Virtual Hypersphere Partitioning. *PVLDB*, 13(9): 1443-1455, 2020.
DOI: https://doi.org/10.14778/3397230.3397240

[*]Corresponding author

Proceedings of the VLDB Endowment, Vol. 13, No. 9
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3397230.3397240

1. INTRODUCTION

The nearest neighbor (NN) search in the high dimensional Euclidean space is of great importance in areas such as database, information retrieval, data mining, pattern recognition and machine learning [2, 18]. To remove the curse of dimensionality, the common wisdom is to design efficient c-approximate NN search algorithms by trading precision for speed [6]. A point o is called a c-approximate NN (c-ANN) of 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 [16], and finds broad applications in practice [1, 18].

As a matter of fact, the original LSH method (E2LSH) does not support c-ANN search directly and a naive extension may cause prohibitively large storage cost [29]. To this end, several LSH variants such as LSH-forest [29], C2LSH [12] and QALSH [15], have been proposed in order to answer c-ANN queries with reasonably small indexes, constant success probability and sub-linear query overhead. We focus on QALSH next since it is more efficient compared with LSH-forest and C2LSH, and closely related to our proposal.

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.

As will be discussed in Section 4, this technique essentially examines points within an *irregular and unbounded region* in *the original feature space.* By statistical analysis, we observed that points with large collision numbers may be less likely to be NNs if their projection distances to q are very large (areas that are inside the crossroad-like region and outside the circle in Figure 1), whereas points with small collision numbers may be promising ones if they are rather close to the query in the projection space (red area in Figure 1). This suggests that the collision-threshold-based filtering may miss promising points and check unfavorable ones.

Motivated by this observation, we introduce the concept of *virtual hypersphere partitioning (VHP)*. As illustrated in Figure 1, the core idea is to impose a virtual hypersphere of radius l, centered at query, in *the original feature space*. For each o, we estimate its distance to q, denoted by $\tilde{d}(o,q)$, in the original space using the LSH projection information. If $\tilde{d}(o,q) \leq l$, i.e., o falls inside the hypersphere, we say o collides with q and put o into the candidate set for further examination. Because the search space is isotropic and bounded, virtual hypersphere partitioning is potentially more efficient than the collision-threshold-based filtering.

Since it is difficult to construct a real hypersphere directly in the original feature space, we developed a principled method to emulate a single virtual hypersphere by using m physical hyperspheres in different projection subspaces in this paper. The working mechanism and soundness of our proposal will be discussed in detail in Section 5.

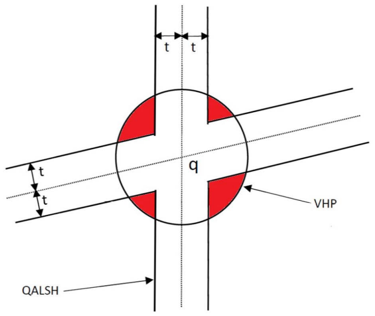


Figure 1: An illustrative example of the search spaces of virtual sphere partitioning and collision-threshold-based filtering. The dimensionality d of the feature space is 2, the number of hash functions m is 2 and the collision threshold L is set to 1. The search window is of size $2t$. The crossroad-like 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.

Using virtual hypersphere partitioning, we proposed an efficient c-ANN algorithm named VHP for searching disk-based large datasets. In the indexing phase, VHP stores LSH projections with independent B^+-trees. To process a query, VHP keeps increasing the radii of physical hyperspheres coordinately, which in effect amounts to enlarging the virtual hypersphere, to accommodate more candidates until the success probability is met. Rigorous theoretical analysis shows that the proposed algorithm supports c-ANN search for arbitrarily small $c \geq 1$ with probability guarantee. Extensive experiments on a variety of datasets, including the billion-scale ones, demonstrate that VHP is a preferable c-ANN search algorithm.

Our main contributions are summarized as follows.

- We introduce the concept of virtual hypersphere partitioning, which uses the estimated distance of o to q as a more efficient indicator to distinguish relevant and irrelevant points in the pruning phase. The soundness is proved based on the solid estimation theory.
- We proposed an efficient disk-based c-ANN search algorithm VHP, which is guaranteed in supporting c-ANN search for arbitrary $c \geq 1$ with user-specified success probability.
- Extensive experiments show that VHP offers desirable recalls for a variety of real datasets with different sizes and distributions, and achieves up to 4x speedup over the state-of-the-art algorithms.

The rest of this paper is organized as follows: Preliminaries are presented in Section 2. The related work is overviewed in Section 3. Section 4 outlines the limitations of QALSH. Section 5 explains the basic idea of VHP and presents the algorithm. The theoretical analysis is presented in Sections 6 and 7. Section 8 lists some discussions about VHP. The experimental comparison is shown in Section 9. Section 10 concludes the paper.

2. PRELIMINARIES

In this paper, 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 $\vec{a} = [a_1,a_2,\cdots,a_d]$ 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 [9].

Lemma 1. *For any points o_1, o_2 in $\Re^d$, $h(o_1)-h(o_2)$ 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 (1) [15].

$$p(s) = Pr[\delta(o) \leq t] = \int_{-\frac{t}{s}}^{\frac{t}{s}} \varphi(x)dx, \qquad (1)$$

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

According to Equation (1), it is easy to see that $p(s)$ is a monotonically decreasing function for fixed t. This means that the probability that q and o fall into the same search window decreases with their Euclidian distance. Based on the definition of locality sensitive hashing, one can prove that $h(\cdot)$ is the query-aware LSH family [15].

Suppose X follows the normal distribution $\mathcal{N}(\mu, \sigma^2)$ and lies within the interval $X \in [a, b],\ -\infty \le a < b \le \infty$. Then X conditional on $a \le X \le b$ 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:

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

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

Truncated normal distribution is important for our proposal 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 $\mathcal{N}(0, d^2(o_1, o_2))$.

Table 1 summarizes the notations that are frequently used in this paper, where the precise explanations of some notations will be deferred to Section 5.

Table 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 $2t$.
$\mathcal{I}_t(o)$	The set of hash functions satisfying $\lvert h_i(q) - h_i(o)\rvert \le t$.
$r_t(o)$	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.

3. RELATED WORK

NN search in high dimensional spaces is a challenging problem that has been studied extensively in the past two decades [11, 19, 20, 22, 29, 32, 33]. There is extensive work on hashing for similarity search in high dimensional spaces, as discussed in literature surveys [1, 31] and empirical comparison [4, 28]. A recent survey of tree-based NN search algorithms can be found in [27].

3.1 LSH-based Algorithms

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 to arbitrarily bad outputs, or uses multiple hashing tables tailored for different radii, which may lead to prohibitively large space consumption in indexing. To reduce the storage cost, LSB-Forest [29] and C2LSH [12] use the so-called virtual rehashing technique, implicitly or explicitly, to avoid building physical hash tables for each search radius.

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 $m = 6$) to store the $\langle g(o), oid\rangle$ 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 in the ascending order of their $\Delta_m(o)$.

To achieve better efficiency, many LSH extensions such as Multi-probe LSH [24], SK-LSH [23], LSH-forest [5] and Selective hashing [14] use heuristics to access more plausible buckets or re-organize datasets, and do not ensure any LSH-like theoretical guarantee.

3.2 Non-LSH Algorithms

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 [13]. Production quantization (PQ) divides the feature space into disjoint subspaces and then quantizes each subspace separately into multiple clusters [17]. 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 [31], 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 [26] 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 [25]. 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 [3]. HD-index [3] is proposed to support the approximate NN search for disk-based billion-scale datasets, which consists of a set of hierarchical structures called RDB-trees built on Hilbert keys of database objects.

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 [7]. While both VHP and PAC-NN use hyperspheres, there are two fundamental differences between them: (1) PAC-NN uses a single physical hypersphere in the original space directly, whereas VHP employs multiples physical hyperspheres in projection subspaces to emulate one virtual hypersphere 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-style algorithm, VHP has no assumption on data distribution, and thus is data independent.

A recent experimental study extends some data-series algorithms, i.e., iSAX2+ and DSTree, to support PAC (probably approximately correct) NN query [10]. The extension is based on the idea of PAC-NN [7], which makes the PAC iSAX2+ and DSTree data dependent in answering probalistic δ-ϵ-approximate queries when $\delta < 1$. As a result, it is reported that they demonstrate better accuracy and efficiency than SRS and QALSH.

4. LIMITATIONS OF QALSH

In this section, we discuss how QALSH works and its limitations from a geometric point of view.

4.1 Brief Review

As discussed in Section 1, 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 (3).

$$|\{i, 1 \leq i \leq m \mid |h_i(o) - h_i(q)| \leq t\}| \geq L \qquad (3)$$

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, 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.

4.2 Geometric Interpretation and Its Limitations

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 defined by j

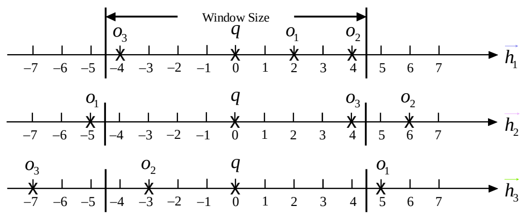


Figure 2: A running example. Three hash functions are used and the search window is of size 9. Three points o_1, o_2 and o_3 are mapped into different projection subspaces.

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 1, the crossroad-like region depicts the search space of QALSH in the case of $d = m = 2$ and $L = 1$, where the solid line represents a degraded hyperplane in the 2-dimensional space. It is easy to see that the search strategy of QALSH has two 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 in the following section.

5. 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 5.1 and Section 5.2, respectively. The detailed algorithm is described in Section 5.3.

5.1 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 1, 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 $\Re^d$ into the m-dimensional projection space $\Re^m$. Due to the existence of search window $[-t, t]$, a point may lie in a query-centric i-constrained projection subspace, which is defined as follows.

DEFINITION 1. *A query-centric i-constrained projection subspace is composed of $x \in \Re^m$ 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 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 6.2, 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 (4) 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.

$$\bigvee_i \{r_t(o) = i \wedge \Delta^t(o) \leq l_i\}, 1 \leq i \leq m \tag{4}$$

It is easy to see that Equation (4) is more stringent and will degrade to Inequality (3) 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 4 as a filtering condition is like examining points whose exact distances to q (in estimation) are less than the virtual radius.

5.2 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 3.

Before query processing, we need to set proper l_i to guarantee the result quality. As will be discussed in Section 6.3, 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 3(a) and Figure 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 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 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 3(b)).

Figure 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 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 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 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 6.4 and Section 7.1 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.

5.3 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 as an example, $r_{4.5}(o_2) = 2$ and $\Delta^{4.5}(o_2) = 5$.

We present the probabilistic NN version of VHP in Algorithm 1, while leaving the c-k-ANN version to Section 7.2. 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 $l_{r_t(o)}^t = \frac{t}{t_0} l_{r_t(o)}^{t_0}$. 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).

[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.

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

```
Input: $q$ is the query point; $2t_0$ 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 $d(o_{\min}, q) > \frac{t}{t_0}$ 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 $\Delta^t(o) \le \frac{t}{t_0} l_{r_t(o)}^{t_0}$ then
6         calculate $d(o, q)$;
7         update $o_{\min}$ if necessary;
8 return $o_{\min}$
```

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.

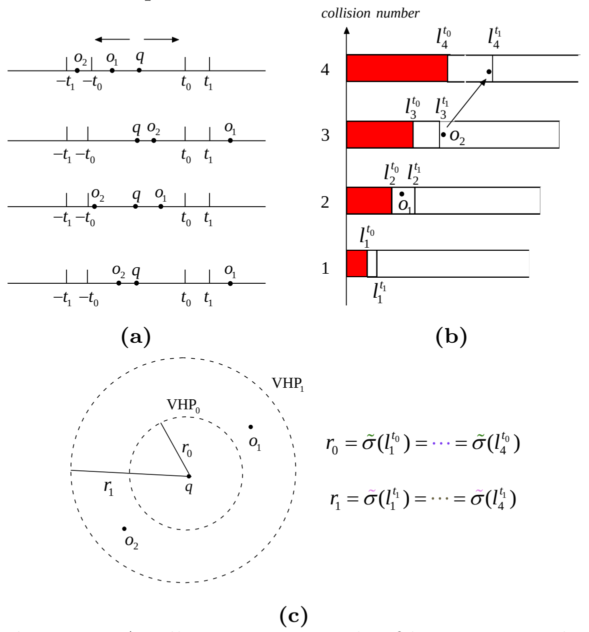


Figure 3: An illustrative example of how VHP works.

6. DETERMINE THE RADII OF PHYSICAL HYPERSPHERES

In this section, we will discuss how to determine the radii of physical hyperspheres. The collision probability between two points is derived in Section 6.1. The method to estimate the virtual radius for one physical hypersphere is discussed in Section 6.2 and the soundness of virtual hypersphere partitioning is shown in Section 6.3. The way to calculate the base hypersphere radii and the practical termination condition are presented in Section 6.4.

6.1 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 $\Delta^t(o) \le l_{r_t(o)}$ and B denote the event $r_t(o) = i$ ($1 \le i \le m$), 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.

$$p_{\mathcal{L}}^t(s) = \sum_{i=1}^{m} C(m, i)(p(s))^i(1 - p(s))^{m-i} \cdot \mathcal{G}_i^t(l_i; 0, s^2) \quad (5)$$

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:

$$p_{\mathcal{L}}^t(s_*) = P_* \quad (6)$$

There may exist many $\mathcal{L}$'s that make Equation (6) 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.

6.2 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 $t_1 \le x \le t_2$ and a *censored observation* otherwise, where t_1 and t_2 are two censoring points [8]. 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 $x_j, 1 \le j \le m$ 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 [8] 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

$$L(\sigma) = [\Phi(t; 0, \sigma^2)]^{m-i} \cdot \prod_{j=1}^{i} \varphi(x_j; 0, \sigma^2) \tag{7}$$

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 $\xi = -t/\sigma$.

$$\frac{\partial(\ln L(\sigma))}{\partial\sigma} = -\frac{(m-i)\varphi(\xi; 0, \sigma^2)}{\sigma\Phi(\xi; 0, \sigma^2)} - \frac{i}{\sigma} + \frac{1}{\sigma^3}\sum_{j=1}^{i} x_j^2 = 0 \tag{8}$$

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 (8) 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 [8].

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 (8) into the following equation, where $G(i, \xi) = [\xi \cdot \frac{m-i}{i}) \cdot \varphi(\xi; 0, \sigma^2) + \Phi(\xi; 0, \sigma^2)]/[\xi^2 \cdot \Phi(\xi; 0, \sigma^2)]$.

$$\|x_j\|^2 = it^2 G(i, \xi) \tag{9}$$

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 (9), and 2) $\tilde{\sigma}(\|x_j\|)$ increases monotonically with $\|x_j\|$ for given m, i and t.

6.3 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.

$$\tilde{\sigma}(l_i) = \tilde{\sigma}(l_j), 1 \le i, j \le m \tag{10}$$

Next we will show that, given P_*, m, t and s, Equation (10) along with Equation (6) 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 $(1 \le j \le i)$ 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 (8), 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 $w \le m - i$.*

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 $|x_j| \le t$, we have

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

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 (10), we have $l_i < l_j$ for any $1 \le i < j \le m$.*

Now we are in the position to show the uniqueness of $(l_1, l_2, ..., l_m)$ under the constraints of Equation (10) and Equation (6).

Theorem 1. *Given P_*, m, t and s, there exists a unique sequence $(l_1, l_2, ..., l_m)$ such that Equation (10) and Equation (6) hold.*

Proof (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 $\Delta^t(o) \le l_i$ for any $1 \le i \le m$ (Equation 4) is equivalent to $\tilde{\sigma}(\Delta^t(o)) \le \tilde{\sigma}(l_i)$ readily, which justifies the soundness of virtual hypersphere partitioning.

6.4 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 $\Delta^t(o) \le l_{r_t(o)}$, 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 $t = t_0$. $\square$

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 ($sl_1^{t_0}$, $sl_2^{t_0}$,..., $sl_m^{t_0}$) 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,..., 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 $\Delta^t(o) \le l_{r_t(o)}$, VHP maintains o_{min}, which is the nearest point to q found so far. As will be proved in Section 7.1, 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 $p_{\mathcal{L}}^{d(q,o_{min})t_0}(d(q, o_{min})) \ge P_*$ 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 (11) as the termination condition, which is much cheaper to evaluate.

$$d(q, o_{min}) \ge \frac{t}{t_0} \tag{11}$$

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 (6) and Equation (10) to get a unique
  solution $(l_1^{t_0}, l_2^{t_0}, ..., l_m^{t_0})$;
2 return $(l_1^{t_0}, l_2^{t_0}, ..., l_m^{t_0})$
```

The equivalency between Inequality (11) and $p_{\mathcal{L}}^{d(q,o_{min})t_0}$ $(d(q, o_{min})) \ge P_*$ can be proved readily using Proposition 2.

The following proposition, which can be proved using Equation (9), shows that the radius of the virtual hypersphere is still unique after scaling.

Proposition 3. *For any $1 \le i, j \le m$ 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 $\tilde{\sigma}(l_i^{st_0}) = s\tilde{\sigma}(l_i^{t_0})$.*

7. THEORETICAL ANALYSIS

7.1 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 $s_a \ge s_*$, we have $t_a \ge t_*$ and $l_i^{t_a} \ge l_i^{t_*}$. 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 $\Delta^{t_2}(o) \le \frac{t_2}{t_0} l_{r_{t_2}(o)}^{t_0}$ if $\Delta^{t_1}(o) \le \frac{t_1}{t_0} l_{r_{t_1}(o)}^{t_0}$.*

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

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 $\tilde{\sigma}(\Delta^{t_2}(o_\dagger)) \le \frac{t_2}{t_1}\tilde{\sigma}(l_{r_{t_1}(o)}^{t_1})$ by Equation (9). Then, with the help of Lemma 3 we know that $\tilde{\sigma}(\Delta^{t_2}(o_\ddagger)) \le \tilde{\sigma}(\Delta^{t_2}(o_\dagger))$ since $(\Delta^{t_2}(o_\dagger))^2 + wt_2^2 \ge (\Delta^{t_2}(o_\ddagger))^2$. Recall that $\tilde{\sigma}(\Delta^t(o))$ is an increasing function of $\Delta^t(o)$ for fixed t and i, and thus we have $\tilde{\sigma}(\Delta^{t_2}(o)) \le \tilde{\sigma}(\Delta(^{t_2}o_\ddagger))$ considering $\Delta^{t_2}(o) \le \Delta^{t_2}(o_\ddagger)$. By putting these inequalities together, it holds that $\tilde{\sigma}(\Delta^{t_2}(o)) \le \frac{t_2}{t_1}\tilde{\sigma}(l_{r^{t_1}(o)}^{t_1})$. According to Proposition 3, we have $\frac{t_2}{t_1}\tilde{\sigma}(l_{r^{t_1}(o)}^{t_1}) = \tilde{\sigma}(l_{r^{t_2}(o)}^{t_2})$, thus complete this proof. $\square$

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 $t_2 \ge t_1$. 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. □

7.2 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. □

8. DISCUSSION

In the worst case, the time complexity of VHP is $O(n(m+d))$. As will be discussed in Section 9, 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 9, 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 $\{l_i^{t_0}\}$.

9. EXPERIMENTAL VALIDATION

In this section, we study the performance of VHP using six real datasets of various size and dimensionality. For comparison, we choose QALSH and SRS as the baseline algorithms because they are state-of-the-art methods that fall into the same category with our proposal, i.e., disk-based indexing techniques that support c-ANN search for large datasets with theoretical guarantee. In addition, we compared VHP with HD-index, a state-of-the-art non-LSH external memory algorithm for billion-scale ANN search.

Our method is implemented in C++. All the experiments were carried on a PC with Intel(R), 3.40GHz i7-4770 eight-cores processor and 8 GB RAM, in Ubuntu 16.04.

9.1 Experiment Setup

9.1.1 Benchmark methods

- **SRS [28].** 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 [21, 28] and P_* was set to 0.9 by default.

- **QALSH [15].** The success probability of QALSH was set to $(1/2 - 1/e)$, as suggested in [15]. m was computed using the method described in [15] as well. c was set to 2 by default unless stated otherwise.

- **HD-index [3].** 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 [3].

- **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 9.1.4, 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.

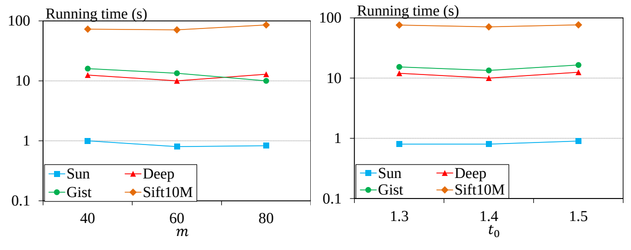
For all methods, we used their external-memory versions. To be specific, for SRS we use the R-tree based external-memory version, which is originally proposed to support billion scale datasets on a commodity PC [28]. As for HD-index and VHP, we build RDB-tree/B+-tree by bulk loading such that they can scale in our setting.

9.1.2 Datasets

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.

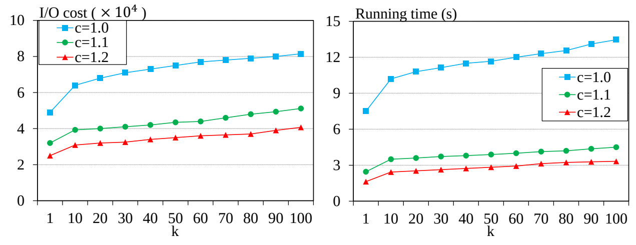
- **Sun**[2] consists of GIST features of images.

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



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

Figure 4: The impact of different parameters



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

Figure 5: The performance of VHP under different approximation ratios

Table 2: Dataset Statistics

Dataset	Dimensionality	Size	Page size
Sun	512	79,106	4KB
Deep	256	1,000,000	4KB
Gist	960	1,000,000	4KB
Sift10M	128	11,164,666	4KB
Sift1B	128	999,494,170	4KB
Deep1B	96	1,000,000,000	4KB

- **Deep**[3] contains deep neural codes of natural images obtained from the activations of a convolutional neural network.
- **Gist**[4] is an image dataset which contains about 1 million data points.
- **Sift10M**[5] consists of 10 million 128-dim SIFT vectors.
- **Sift1B**[6] consist of 1 billion 128-dim SIFT vectors.
- **Deep1B**[7] consist of 1 billion 96-dim DEEP vectors.

9.1.3 *Evaluation metrics*

We consider the following four performance metrics similar to [15, 21, 28]:

- **I/O cost.** I/O cost, which denotes the number of pages accessed, is an important metric for external memory algorithms. I/O costs consist of the overhead in both index and data access. For the fairness, only identified candidates will be accessed during the search.
- **Running time.** The running time for processing a query is also considered. It is defined as the wallclock time for a method to solve the c-k-ANN problem.

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

Table 3: 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

- **Overall ratio.** Overall ratio is used to measure the accuracy of these algorithms. For c-k-ANN search, the overall ratio is defined as $\frac{1}{k}\sum_{i=1}^{k}\frac{d(o_i,q)}{d(o_{*i},q)}$.
- **Recall.** Recall is used as the other 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.

It is notable that, in some recent papers such as [3], *Mean Average Precision(mAP)* is also used as an important performance metric. In this paper, since all methods adopt the filter-and-verify strategy and return objects in the ascending order of their exact distances to queries, their mAPs are exactly equal to the corresponding recalls.

9.1.4 *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 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.

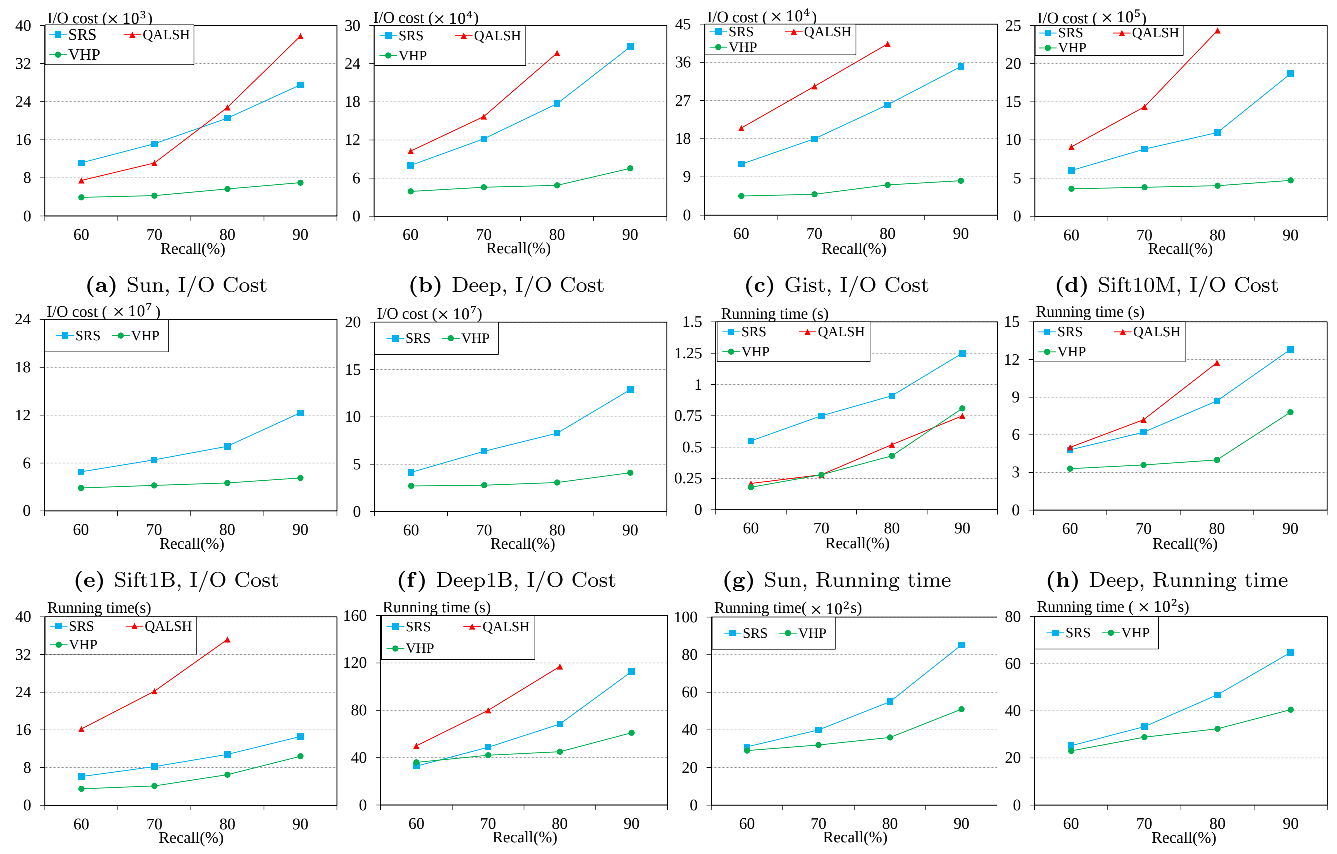
(2) As for m, we can see that VHP works well when $m = 60$. It is notable that the performance of VHP on Gist can be better in the case of $m = 80$. This is because the dimensionality of Gist is much higher (dim 960) and more hash functions may help to distinguish nearest neighbors better.

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

9.2 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 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



(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 6: The comparison on the accuracy-efficiency tradeoffs of VHP, SRS and QALSH

using the worst-case analysis, which is often not the case of real datasets.

9.3 Index Size, Indexing Time and Memory Consumption

Table 3 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.

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.

9.4 VHP vs. LSH-based Methods

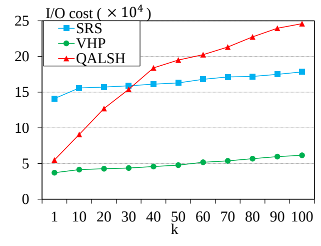
In this section, we compare VHP with the baseline LSH-based methods SRS and QALSH. In order to make the comparison more reasonable, we fix the expected recall and measure how much running time and I/O cost it takes for three LSH-based methods in this paper. Such a comparison is feasible because all of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

9.4.1 Experimental results under the same recall

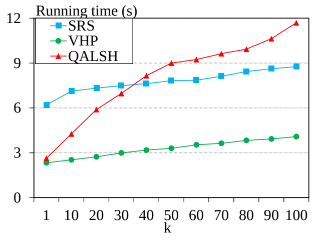
In Figure 6, the target recalls are set to 60%, 70%, 80%, 90% because they are high enough for the practical use. According to the results, we have following observations.

(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.

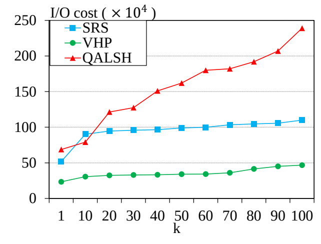
(2) The superiority of VHP over the other two methods becomes relatively less significant at low recall, say 60%. This is because, as the target recall is getting lower, it becomes easier for all three methods to find answers satisfying the less strict requirement, which in turn reduces the differences 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.



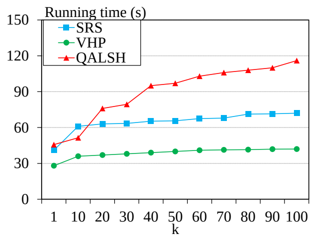
(a) Deep, I/O Cost



(b) Deep, Running time



(c) Sift10M, I/O Cost



(d) Sift10M, Running time

Figure 7: The performances of VHP under different k at recall 80%

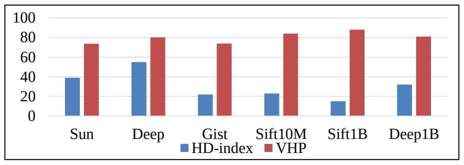
9.4.2 *Experimental results under different k*

The results above were all obtained under $k = 100$, we also compared three methods for different k in Figure 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.

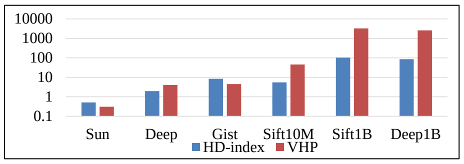
9.5 VHP vs. HD-index

In this section, VHP is compared with HD-index, the state-of-the-art non-LSH (without probability guarantees) ANN search algorithm for large disk-based dataset. HD-index cannot solve c-ANN search problem and does not collect the statistics about the I/O cost, thus we only evaluate the recall and running time to measure the accuracy and efficiency. Figure 8 lists the experimental results for VHP and HD-index. For VHP, c was set to 1.1 and $k = 100$. For HD-index, the optimal parameters were used, by which the best performance could be achieved [3]. A few interesting observations are made as follows.

(1) Regardless of the data types and distributions, VHP constantly delivers satisfactory recalls (78%-85%), which demonstrates again the advantage of ANN search algorithms with theoretical guarantee. In contrast, the accuracy of HD-index varies significantly for different datasets. Specifically, HD-index reaches the minimum recall of 13% for Sift1B and the maximum recall of 55% for Deep. Such unpredictability makes it hard to meet the expectation of end users in practice. Note that even laborious parameter tuning cannot help here - the parameters of HD-index have been adjusted to be experimentally optimal and the accuracy could not be improved by tuning internal parameters as reported in [3].



(a) Recall(%)



(b) Running time(s)

Figure 8: HD-index vs. VHP ($c = 1.1$), $k = 100$

(2) The accuracy of HD-index gets lower on high-dimensional or large datasets (Gist, Sift1B). One possible reason is that HD-index uses heuristics such as filling curve to identify NN. Since the filling curve is one-dimensional, many true kNNs are far away form the query on each RDB-tree due to the so-called "boundary effect" [30], which results in that HD-index could not achieve high accuracy even if it is allowed a larger search range as admitted in [3]. In contrast, VHP uses the virtual hypersphere partitioning to selectively examine those points with high probability being the NN of the query. Hence, even for datasets with high dimensionality and/or large size, VHP can still achieve much higher accuracy than HD-index.

(3) As for efficiency, we can see that, in most cases, HD-index spends less running time than VHP, especially for large datasets. This, however, may not necessarily mean that VHP is less efficient since the recalls of VHP are far higher than those of HD-index, not mentioning its salient feature in ensuring the answer quality. One interesting observation is that VHP outperforms HD-index in both recall and running time on those datasets with higher dimensions (Sun and Gist).

In short, VHP is more preferable than HD-index in situations where the answer quality and stability are of great importance.

10. CONCLUSION

The nearest neighbor search in high dimensional spaces is a difficult problem. In this paper, we propose a novel approximate NN search algorithm called VHP. VHP works with arbitrarily small approximation ratio $c \geq 1$ and is guaranteed to identify c-k-ANN with the given success probability, which is of great practical importance. Compared with existing methods over large real datasets, VHP achieves better efficiency under the same answer quality.

Acknowledgment

The work reported in this paper is partially supported by NSFC under grant number 61370205, NSF of Xinjiang Key Laboratory under grant number 2019D04024 and JSPS KAKENHI under grant number 19H04128. Wei Wang were supported by ARC DPs 170103710 and 180103411, and D2DCRC DC25002 and DC25003.

11. REFERENCES

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

[2] 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.

[3] 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.

[4] M. Aumüller, E. Bernhardsson, and A. Faithfull. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. In *SISAP*, pages 34–49, 2017.

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

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

[7] 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.

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

[9] 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.

[10] 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.

[11] R. Fagin, R. Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In *SIGMOD*, pages 301–312, 2003.

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

[13] 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.

[14] 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.

[15] 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.

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

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

[18] 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.

[19] Y. Lei, Q. Huang, M. Kankanhalli, and A. Tung. Sublinear time nearest neighbor search over generalized weighted space. In *ICML*, pages 3773–3781, 2019.

[20] M. Li, Y. Zhang, Y. Sun, W. Wang, I. W. Tsang, and X. Lin. I/O efficient approximate nearest neighbour search based on learned functions. In *ICDE*, 2020.

[21] 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.

[22] W. Liu, H. Wang, Y. Zhang, W. Wang, and L. Qin. I-LSH: I/O efficient c-approximate nearest neighbor search in high-dimensional space. In *ICDE*, pages 1670–1673, 2019.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] P. Ram and K. Sinha. Revisiting kd-tree for nearest neighbor search. In *KDD*, pages 1378–1388, 2019.

[28] 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.

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

[30] E. Valle, M. Cord, and S. Philipp-Foliguet. High-dimensional descriptor indexing for large multimedia databases. In *CIKM*, pages 739–748, 2008.

[31] 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.

[32] 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.

[33] B. Zheng, X. Zhao, L. Weng, N. Q. V. Hung, H. Liu, and C. S. Jensen. PM-LSH: A fast and accurate LSH framework for high-dimensional approximate NN search. *PVLDB*, 13(5):643–655, 2020.