DASH: Data Aware Locality Sensitive Hashing

Zongyuan Tan[1,3], Hongya Wang[1,2,3(✉)], Ming Du[1], and Jie Zhang[1]

[1] School of Computer Science and Technology, Donghua University, Shanghai, China
tanzongyuan@mail.dhu.edu.cn, hywang@dhu.edu.cn
[2] State Key Laboratory of Computer Architecture, ICT, CAS, Beijing, China
[3] Shanghai Key Laboratory of Computer Software Evaluating and Testing, Shanghai, China

Abstract. Locality sensitive hashing (LSH) has been extensively employed to solve the problem of c-approximate nearest neighbor search (c-ANNS) in high-dimensional spaces. However, the search performance of LSH is degenerated with the number of data increasing. To this end, we propose an efficient method called *Data Aware Sensitive Hashing* (DASH) to deal with this drawback. DASH is the data-dependent hashing algorithm under considering the residual distance prior. DASH leverages this prior knowledge and provides theoretical guarantee for search results. Our experimental results with various datasets show that DASH achieves better search performance and the running time can reach up to about 4–40x speedups compared with other state-of-the-art methods.

Keywords: LSH · ANNS · High dimensions · Data-dependent hashing

1 Introduction

The nearest neighbor search (NNS) is the research focus all the time, which has been extensively applied to various fields, such as databases, machine learning and data mining. Given a query point q with dimension d in the Euclidean space, the problem of NNS is to return a point o^* in the dataset $\mathcal{D}$ with minimum distance to q. For low-dimensional NNS, the exact solutions have already been reported by based-tree methods. However, it has a great challenge to find the exact results for NNS in high-dimensional space due to the curse of dimensionality. Hence, an alternative scheme, i.e., the approximate nearest neighbor search (ANNS), has been extensively studied in recent two decades. Formally, the purpose of c-ANNS is to report a point $o \in \mathcal{D}$, whose distance with q is within $c \times r^*$, where r^* represents the distance between the query q and its exact nearest neighbor.

Locality sensitive hashing (LSH) is one of most effective techniques to solve high-dimensional c-ANNS problems, which is originally proposed for hamming space in [1], and later is extended to Euclidean space based on p-stable distribution [2]. Since LSH often needs to build hundreds of hash tables for achieving

B. Li et al. (Eds.): APWeb-WAIM 2022, LNCS 13422, pp. 85–100, 2023.
https://doi.org/10.1007/978-3-031-25198-6_7

good search results. This leads to prohibitively large space consumption. To deal with this problem, many LSH variants have been designed, e.g. [3–5]. However, LSH and its variants are data-independent hashing schemes.

As a matter of fact, many researchers turn to study the data-dependent hashing schemes to enhance the search performance for ANNS. In [7], the best LSH data structure is constructed by partitioning the original datasets into several subsets to form data-dependent hashing scheme. Based on [7], another data-dependent hashing scheme is proposed via spectral theorem [8]. Although the both methods have rigorous theoretical guarantee for search results returned, it is hard to put them to practice. Moreover, DSH [9] improves the hashing functions to address the problem that the elements of buckets for traditional LSH are unbalance. But DSH is lack of similar probability guarantee with traditional LSH. Our proposal DASH is the data-dependent hashing algorithm under considering the residual distance prior, and has the nature of probability guarantee for LSH. Similarly, BayesLSH [23] also puts forward a prior distribution and exploits Bayes inference to give the probability guarantee for the results returned, while its prior distribution is data-independent.

Motivations. Most of LSH and its variants have desirable theoretical guarantee for the search results, but often suffer from time inefficient, although they can achieve answering c-ANN queries with sub-linear query overhead. Also, the calculation cost on Euclidean distance between the query and its candidates is great because of finding a large amount of useless candidates. Product quantization (PQ) [10] provides effective means to estimate Euclidean distance for any high-dimensional points in Euclidean space. It constructs pre-calculation distance table by the manner of the asymmetric distance computation (ADC) or symmetric distance computation (SDC). This speeds up the distance computation for any two points, compared against computing Euclidean distance directly. By taking the merit of LSH and PQ into account, we are expected to design an algorithm that not only has probability guarantee for c-ANNS, but also is able to speed up query processing.

Contributions. The main contributions of the paper are concluded as follow:

- we propose an algorithm called Data Aware Sensitive Hashing (DASH) to answer the k-ANNS in high-dimensional Euclidean space. DASH is time efficient method and provides quality guarantee for the search results returned with preassigned success probability.
- We propose a novel prior (residual distance prior) – the key observation for DASH, which is based on the statistics of residual distance on data points to any random query. Equipped with this prior knowledge, DASH is able to address the k-NNS problem in more efficient manner.
- Extensive experiments demonstrate that DASH achieves desirable search performance for a variety of real datasets with different sizes. Compared against other state-of-the-art algorithms, DASH can obtain at least 4x speedup in the running time over different datasets.

Organization. The rest of this paper is organized as follows. Some preliminaries are reviewed in Sect. 2. A key observation is found in Sect. 3. Our method and probability analysis are presented in Sect. 4. Experimental results and analysis are reported in Sect. 5. Section 6 discusses the related work. Finally, a brief conclusion is drawn in Sect. 7.

2 Preliminaries

2.1 Problem Definitions

In this paper, we consider a dataset $\mathcal{D}$ with n points denoted as vectors in a d-dimensional Euclidean space R^d. For any point o and query q, let $d(o, q)$ represent the Euclidean distance that is defined as $d(o, q) = \sqrt{\sum_i^d (o[i] - q[i])^2}$, where $o[i]$ and $q[i]$ are the coordinate value of i-th dimension for o and q, respectively. Given a query q and the distance measure $d(\cdot, \cdot)$, the exact nearest neighbor (NN) o^* of q is the point in $\mathcal{D}$ with the minimum distance to q, namely $o^* = argmin_{o \in \mathcal{D}} d(o, q)$. Then, the c-approximate nearest neighbor search (c-ANNS) is defined as follows:

Definition 1. *Given an approximate ratio c ($c \geq 1$), any query $q \in R^d$ and the distance measure $d(\cdot, \cdot)$, the problem of c-ANNS is to establish a data structure, which retrieves a point $o \in \mathcal{D}$ satisfying $d(o, q) \leq c \times d(o^*, q)$, in which $o^* \in \mathcal{D}$ denotes the exact NN of q.*

The c-ANNS can be extended to more generalized form of c-k-ANNS. Similarly, the problem of c-k-ANNS is to establish a data structure, which for any query $q \in R^d$, retrieves a set of k ordered points $o_i \in \mathcal{D}$ ($1 \leq i \leq k$) satisfying $d(o_i, q) \leq c \times d(o_i^*, q)$, in which $o_i^* \in \mathcal{D}$ denotes the i-th exact NN of q.

2.2 Product Quantization

Product Quantization (PQ) [10] has been proposed to address the ANNS problem in high-dimensional space. It divides the d-dimensional original space into M subspaces equally, with the dimension of each subspace being $\frac{d}{M}$. Correspondingly, all the original vectors are divided into M subvectors and the dimension of each subvector is $\frac{d}{M}$. Then all the subvectors in each subspace are quantized to k^* different centroids, which are learned from a part of original data by k-means algorithm. That is, each subvector is denoted by the index of its nearest centroid, where the index is an integer over interval $[1, k^*]$. Thus, each original vector is denoted by a tuple of M integers, which are called as PQ-code. All the $M \cdot k^*$ centroids are compose of the codebook C of the product quantizer jointly.

With the codebook and PQ-codes, the Euclidean distance between any two vectors in the original space is estimated from their PQ-codes. There exists two manners to estimate the distance, i.e., the asymmetric distance calculation (ADC, the distance is calculated by a original vector and a PQ-code) and symmetric distance calculation (SDC, the distance is calculated by two PQ-codes). When a query vector arrives, a $M \cdot k^*$ pre-calculation distance table is built with ADC manner.

2.3 Query-Aware LSH Scheme

Assume that there is a random projection vector $\vec{a}$ of dimension d, namely $\vec{a} = [a_1, a_2, \ldots, a_d]$, whose each entry is an i.i.d random variable drawn from Gaussian distribution $\mathcal{N}(0, 1)$, independently. Given a vector $\vec{o}$ of dimension d, the hash function between two vectors $\vec{a}$ and $\vec{o}$ represented as $h(o) = \langle \vec{a}, \vec{o} \rangle$ is the projection of o along $\vec{a}$, which is regarded as an LSH signature. According to the property of p-stable distribution for $p = 2$ [2], for any given points o_1, o_2, $h(o_1) - h(o_2)$ follows Gaussian distribution $\mathcal{N}(0, d^2(o_1, o_2))$.

Given a bucket width $2w$, the strategy of query-aware is that when query point q arrives, its LSH signature is located as the projected centre to identify the interval with bucket width $2w$, i.e., the interval $[h(q) - w, h(q) + w]$. For arbitrary point o, let $s = d(o, q)$, if the LSH signature of a point o falls in the hash bucket with width $2w$, i.e., $|h(o) - h(q)| \leq w$, then o collides with q under the hash function $h(\cdot)$. Accordingly, the collision probability is formalized as following form:

$$p(s) = P_r(\psi(o) \leq w) = \int_{-\frac{w}{s}}^{\frac{w}{s}} \varphi(x)dx \tag{1}$$

with $\psi(o) = |h(o) - h(q)|$ and $\varphi(x)$ being the probability density function of Gaussian Distribution $\mathcal{N}(0, 1)$.

Suppose that the number of independent hash functions and the collision threshold are m and l, respectively, where $l \leq m$. According to the collision counting technique [4], it is easy to derive that the probability $\mathcal{P}(\dagger Col(o) \geq l)$ of point returned obeys Binomial distribution $\mathcal{B}(m, p(s))$, with

$$\mathcal{P}(\dagger Col(o) \geq l) = \sum_{i=l}^{m} \binom{m}{i} (p(s))^i (1 - p(s))^{m-i} \tag{2}$$

in which $\dagger Col(o)$ is the number of collision between q and o under m hash functions.

3 Residual Distance Prior

The residual distance prior is based on the following observation that the difference of approximate distance and Euclidean distance with respect to a random query point forms a specific distribution. To elaborate this observation formally, we first define the notion of residual distance. For a random query point q and an arbitrary point o, their residual distance (denoted by e) is given as following form:

$$e = d(o, q) - \widehat{d}(o, q) \tag{3}$$

where $d(o, q)$ denotes the Euclidean distance between the original vector o and query q, while $\widehat{d}(o, q)$ denotes the approximate distance of $d(o, q)$. Clearly, the residual distance is query-dependent.

Observation. We fit e over different datasets, as shown in Fig. 1 and Fig. 2. One important finding is that e fitted over many existing datasets asymptotically follow a common distribution family, i.e., Gaussian distribution.

Next, we will show how to obtain the universal residual distance distributions. This is similar to the previous works [11]. For a random query point q, we first compute the Euclidean distances $d(o,q)$ and $\widehat{d}(o,q)$ to obtain their difference e. Then the histogram on the difference e forms the residual distance distribution with respect to this query q. Actually, the residual distance distributions constructed for various queries could be discrepant. However, we observe that these residual distance distributions have similar shapes, but their scales are different. If a single residual distribution is only selected in one of the residual distance distributions as an universal distribution, errors will be resulted in. Therefore, in order to reduce error, we select the residual distance distributions for certain queries to approximate the universal residual distance distribution.

Concretely, we show how to extract statistical parameters from various datasets and conduct parameter estimation for the Gaussian distribution. According to the description above, we propose to fit the residual distance by the Gaussian distribution, whose probability density function is formalized as follows:

$$f(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} exp^{\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)} \tag{4}$$

with μ being the shape parameter; σ being the scale parameter. In practice, the purpose we select the Gaussian distribution is that it has distinct advantages.

First, the Gaussian distribution fits the residual distances with respect to various datasets, as shown in Fig. 1 and Fig. 2. Please see Subsect. 5.1 for more details about the datasets. One can be found that the residual distance distributions follow Gaussian distribution asymptotically. Since the accuracy of $\widehat{d}(o,q)$ depends on M, namely the larger M, the more accuracy $\widehat{d}(o,q)$ would be, the residual distance distributions vary with M. To verify the availability, we present different results with the variation of M. In Fig. 1, $\widehat{d}(o,q)$ over various datasets are computed with PQ under $M = 8$, except the dataset ImageNet with $M = 10$. While in Fig. 2, $M = 25$ for ImageNet, and other datasets are $M = 16$. Note that the residual distance distributions are also fitted to be the Gaussian distribution when increase the magnitude of M, where we only show part of the residual distance distributions due to the limitation of space.

Next, there exists an effective method to estimate the parameters of residual distance distribution, which can be estimated by Maximum Likelihood Estimation (MLE). For any sample with n i.i.d. Gaussian random variables, i.e., $\{x_1, x_2, \ldots, x_n\}$, the likelihood function is given by following form:

$$\mathcal{L}(\mu,\sigma) = \prod_{i=1}^{n} f(x_i|u,\sigma^2) \tag{5}$$

then we maximize the log-likelihood function $ln(\mathcal{L}(\mu,\sigma))$. By respectively computing the partial derivatives of $ln(\mathcal{L}(\mu,\sigma))$ for both two parameters μ and σ,

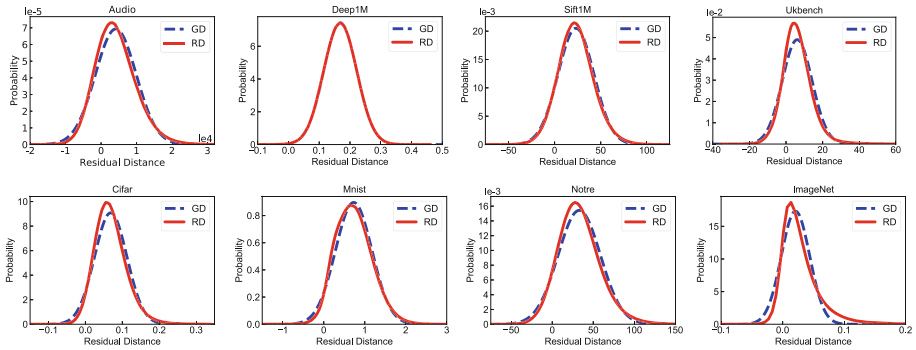


Fig. 1. The residual distance distribution (RD) asymptotically follows the Gaussian distribution (GD) for $M = 8$ and $M = 10$.

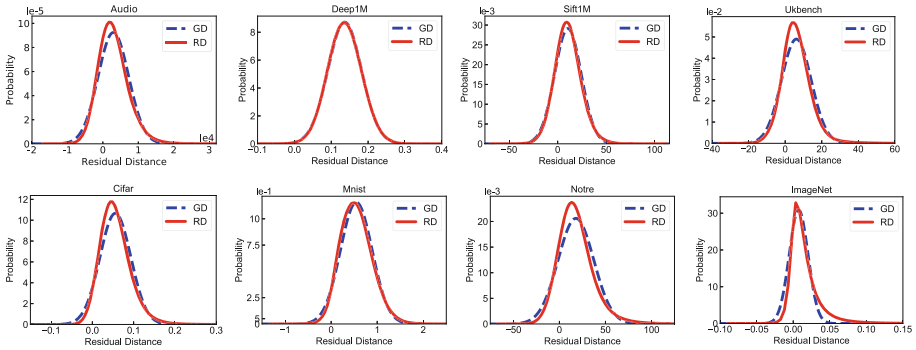


Fig. 2. The residual distance distribution (RD) asymptotically follows the Gaussian distribution (GD) for $M = 16$ and $M = 25$.

their estimation can be solved as $\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i$ and $\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{\mu})^2$. It can be found that the estimated parameters only hinge on arithmetic mean and variance of the sample, such that they can be easily obtained from given dataset.

4 Our Approach and Theoretical Analysis

4.1 Overview

Our algorithm is based on the search framework of QALSH [5]. It achieves that the exact distance calculation between the query and its candidates is converted to a look-up table operation, which greatly speeds up search time. Also, the similar probability guarantee with LSH is still obtained.

4.2 Algorithm Achievement

Our algorithm is based on the framework for memory version of QALSH [5], which can be divided into two parts: indexing construction and query processing.

Indexing Construction. We first select m LSH random projection vectors $\vec{a}$ generated from $\mathcal{N}(0,1)$. Then, each $o \in \mathcal{D}$ is randomly projected from d-dimensional space to m hash values $h_i(o)$, $i \in \{1, 2, \ldots, m\}$. For each projection vector $\vec{a}_i$, we construct a sorted hash table to store the key-value pair $(ID(o), h_i(o))$ for all points, where key $ID(o)$ is the identifier (ID) of each point o, and the hash table is sorted in the ascending order of $h_i(o)$. Finally, m hash tables reside on memory. Meanwhile, we employ PQ quantifying the dataset $\mathcal{D}$ to form PQ-codes and the pre-calculation distance tables for various queries are constructed with ADC manner.

Query Processing. When a query point $q \in R^d$ arrives, we also use m hash functions $h(\cdot)$ mapping it into corresponding hash values $h_i(q)$. Then for given bucket width $2w$, we will conduct a range search $[h_i(q) - w, h_i(q) + w]$ over each hash table. During this search processing, $\dagger Col(o)$ for each point o is updated dynamically. Recall that $\dagger Col(o)$ is the number of collision between o and q.

The pseudo-code of the probabilistic NNS on DASH is shown in Algorithm 1. DASH locates the hash values $h_i(q)$ via binary search, and the range search is conducted by gradually extending bucket width $2w$ by adding $\triangle w$ (Line 3), which is similar to the process of virtual rehashing [4,5] to access more points. However, the most significant difference of extending $2w$ is that it positively impacts the collision probability for supporting our probabilistic terminal condition, which will be discussed in the next subsection. When $|h_i(o) - h_i(q)| \leq w$, $\dagger Col(o)$ is updated (Line 5-6). If $\dagger Col(o)$ is not lower than the collision threshold l, o is regarded as the candidate of q (Line 7-8). Then, the calculation of exact distance $d(o, q)$ is converted to a look-up table operation, i.e., $d(o, q)$ is approximated with $\widehat{d}(o, q)$ calculated by the pre-calculation distance table constructed with PQ. Such an operation accelerates the query processing greatly.

Due to the fact that $\widehat{d}(o, q)$ is an estimation distance that has certain error with $d(o, q)$, DASH is unable to obtain desirable search accuracy under fixed bucket width compared with QALSH. To this end, we supplement the loss $\triangle$ based on $\widehat{d}(o, q)$ to obtain a more accuracy distance, where $\triangle$ is a constant value determined by Gaussian distribution $\mathcal{N}(\hat{\mu}, \hat{\sigma}^2)$, and we will present how to determine $\triangle$ in Subsect. 4.3. Finally, $d(o, q)$ is estimated as $\widehat{d}(o, q) + \triangle$. As $\widehat{d}(o, q) + \triangle$ is large enough, i.e., $\widehat{d}(o, q) + \triangle > d(o, q)$, it leads to DASH extending the bucket width to bring in more points in comparison with QALSH. This because for any given success probability P^*, the bucket width $2w$ is proportional to $\widehat{d}(o, q) + \triangle$. If $\widehat{d}(o, q) + \triangle > d(o, q)$, QALSH and DASH will be terminated early in $2w'$ and $2w''$ respectively, with $w' < w''$. Furthermore, another method is to heighten the success probability P^*. This condition is rigorous, which promotes DASH to return more candidates. Hence, DASH enhances the search accuracy effectively with the methods above.

Algorithm 1: NNS on DASH

Input: query point q, m hash tables, collision threshold l, bucket width $2w$, success probability P^*, approximate ratio c;
Output: o_{min} in the set of $\mathcal{C}$

```
1  w = 0; †Col(o) = 0;
2  while true do
3      2w = 2w + △w;
4      for each i = 1 to m do
5          for |h_i(o) − h_i(q)| ≤ w for o in i-th hash table do
6              †Col(o) = †Col(o) + 1;
7              if †Col(o) ≥ l then
8                  C = C ∪ o;
9                  Calculate (d̂(o,q) + △) and P_r(d̂(o,q) + △);
10                 if P_r(d̂(o,q) + △) ≥ P* then
11                     break the while-loop;
12 return o_min ∈ C;
```

If PQ is directly exploited to the acceleration, it will lead to the destruction on probability guarantee of LSH. Fortunately, we observe that the residual distance prior follows an universal distribution, as described in Sect. 3. According to the key observation, we determine the loss $\triangle$ as a constant value with certain probability. Then by taking this probability into account, we develop a new probability guarantee based on the framework of QALSH as the terminal condition of DASH. Actually the probability guarantee is similar to that in LSH, which is given in Subsect. 4.3. With the estimated distance $\widehat{d}(o,q)+\triangle$, it is easy to derive the collision probability $P_r(\widehat{d}(o,q)+\triangle)$ (Line 9). Assume the terminal condition has already been satisfied (Line 10), o_{min} in the set $\mathcal{C}$ is reported as the final result (Line 12).

***k*-NN Search.** Our method can also be extended to perform k-NN search. It is sufficient to modify the terminal condition as $|\mathcal{C}| \geq k$ and $P_r(\widehat{d}(o,q) + \triangle) \geq P^*$ (Line 10 of Algorithm 1), and finally return k neighbors, i.e., $\{o^1_{min}, o^2_{min}, \ldots, o^k_{min}\}$. Hence, with this search framework, DASH can conduct probabilistic NNS and k-NNS.

4.3 Probability Analysis

Assume that the point o is the candidate of q. As discussed above, their estimated distance is $\widehat{d}(o,q) + \triangle$, and the loss $\triangle$ is determined based on $\mathcal{N}(\hat{\mu}, \hat{\sigma}^2)$. Next, we mainly focus how to obtain $\triangle$ with desirable probability.

According to the $j\sigma$ rule of Gaussian distribution, here $j \in \{1, 2, 3\}$, the probability $\mathcal{P}(\mu, j\sigma)$ for random variables drawn within $[\mu - j\sigma, \mu + j\sigma]$ is given as:

$$\mathcal{P}(\mu, j\sigma) = F(\mu + j\sigma) - F(\mu - j\sigma) = \frac{2}{\sqrt{\pi}} \int_0^{\frac{j}{\sqrt{2}}} dx \tag{6}$$

where $F(\cdot)$ is the cumulative distribution function (CDF) of Gaussian distribution. From the Eq. (3), we have $d(o,q) = \widehat{d}(o,q) + e$, where $e \sim \mathcal{N}(\hat{\mu}, \hat{\sigma}^2)$. Then, one can derive based on Eq. (6) that $d(o,q)$ falls into the interval $[\widehat{d}(o,q) + \hat{\mu} - j\hat{\sigma}, \widehat{d}(o,q) + \hat{\mu} + j\hat{\sigma}]$ with probability $\mathcal{P}(\hat{\mu}, j\hat{\sigma})$. Hence, the loss $\triangle$ is determined as $\triangle = \hat{\mu} + j\hat{\sigma}$, which is the worst-case to estimate $d(o,q)$ under $\mathcal{P}(\hat{\mu}, j\hat{\sigma})$, with $d(o,q) = \widehat{d}(o,q) + \hat{\mu} + j\hat{\sigma}$.

Recall that $\psi(o) = |h(o) - h(q)|$. For given bucket width $2w$ and approximate ratio c, the collision probability $p(\widehat{d}(o,q) + \triangle)$ for both q and o is given as following form:

$$p(\widehat{d}(o,q) + \triangle) = P_r(\psi(o) \leq cw) = \int_{-\frac{cw}{\widehat{d}(o,q)+\triangle}}^{\frac{cw}{\widehat{d}(o,q)+\triangle}} \varphi(x)dx \tag{7}$$

where $\varphi(x)$ is the probability density function of $\mathcal{N}(0,1)$. A simple calculation for the Eq. (7) is $p(\widehat{d}(o,q) + \triangle) = 2norm(\frac{cw}{\widehat{d}(o,q)+\triangle}) - 1$, in which $norm(x) = \int_{-\infty}^{x} \varphi(t)dt$. Note that $norm(x)$ is the CDF of $\mathcal{N}(0,1)$, which is the monotonically increasing function with respect to x. When c and $2w$ are fixed, $norm(\frac{cw}{\widehat{d}(o,q)+\triangle})$ is monotonically decreasing with $\widehat{d}(o,q) + \triangle$ increasing, so $p(\widehat{d}(o,q) + \triangle)$ monotonically decreases with $\widehat{d}(o,q) + \triangle$ increasing. We know that $\widehat{d}(o,q) + \triangle$ is the worse-case to estimate $d(o,q)$. Since $\triangle$ is a random variable drawn from $\mathcal{N}(\hat{\mu}, \hat{\sigma}^2)$ with probability $\mathcal{P}(\mu, j\sigma)$, there is $\widehat{d}(o,q) + \triangle \leq d(o,q)$ or $\widehat{d}(o,q) + \triangle \geq d(o,q)$.

Since $p(\widehat{d}(o,q) + \triangle)$ is known, $\mathcal{P}(\dagger Col(o) \geq l)$ is obtained via the Eq. (2). To achieve the probability guarantee of the algorithm, we first make an assumption that the two events on the loss $\boldsymbol{\Delta}$ determined and the nearest neighbor returned by collision counting are mutually independent under DASH, i.e., the both probability $\mathcal{P}(\dagger Col(o) \geq l)$ and $\mathcal{P}(\mu, j\sigma)$ obtained are independent. Then the overall probability $P_r(\widehat{d}(o,q) + \triangle)$ of finding the nearest neighbor for DASH can be expressed as:

$$P_r(\widehat{d}(o,q) + \triangle) = \mathcal{P}(\dagger Col(o) \geq l) \cdot \mathcal{P}(\mu, j\sigma) \tag{8}$$

This is regarded as the probabilistic terminal condition for DASH. Furthermore, if we could acquire $\widehat{d}(o,q) + \triangle$ in advance, then $p(\widehat{d}(o,q) + \triangle)$ is determined immediately by the Eq. (7). We only require to select suitable m, l and $\mathcal{P}(\mu, j\sigma)$ to realize the success probability P^* specified beforehand, such that

$$P_r(\widehat{d}(o,q) + \triangle) = P^* \tag{9}$$

Example 1. Suppose the point o has been the candidate of q, and their estimated distance is $\widehat{d}(o,q) + \mu + 2\sigma$. This means $\triangle = \mu + 2\sigma$, with $\mathcal{P}(\mu, 2\sigma) = 0.9544$.

If the collision probability $\mathcal{P}(\dagger Col(o) \geq l) = 0.9$, then the overall probability $P_r(\widehat{d}(o,q) + \triangle) = 0.9 \times 0.9544 \approx 0.86$. Similarly, when $d(o,q) = \widehat{d}(o,q) + \mu + 3\sigma$, we obtain $\mathcal{P}(\mu, 3\sigma) = 0.9974$, such that $P_r(\widehat{d}(o,q) + \triangle) = 0.9 \times 0.9974 \approx 0.9$.

To sum up, it is natural to yield the theorems below for the search results under DASH.

Theorem 1. *DASH returns the NN (o_{min}) of query q with the success probability at least P^*.*

Proof. First, we define the two events below:

E_1 : the loss $\triangle$ is determined based on residual distance distribution.

E_2 : the o_{min} is found by DASH.

Recall that we make an assumption on E_1 and E_2 being independent. As discussed earlier, if the points are contained under the fixed bucket width $2w$, then $P[E_1]$ and $P[E_2]$ can be obtained by the Eq. (6) and Eq. (2), respectively. Since the both events are mutually independent, $P[E_1E_2] = P[E_1]P[E_2]$. However, DASH is guaranteed to answer the o_{min} with success probability P^*, then we have $P[E_1E_2] \geq P^*$. Hence, this theorem is proved. □

Theorem 2. *DASH returns the k-NN ($\{o^i_{min}\}_{i=1}^k$) of query q with the success probability at least P^*.*

Proof. The proof of this theorem is similar to Theorem 1. □

5 Performance Evaluation

Our method is implemented in C++ and compiled with g++ 9.3 with -O3 optimization. The experiments for general-scale datasets were conducted on a laptop with six-cores Intel(R), i7-8750H @ 2.20GHz CPUs and 32 GB RAM, in Ubuntu 20.04. While others for large-scale datasets were conducted on a server with eight-cores Intel(R), E5-2620 v4 @2.1GHz CPUs and 256 GB RAM.

5.1 Datasets and Experiment Setting

We employ some publicly available real-life datasets in our experiments, whose data types cover audio, image and deep-learning data. Also, the 50 points are chose randomly from corresponding test sets as queries.

- **Cifar.** The Cifar dataset is a collection of 0.05 million 512-dimensional GIST feature vectors extracted from TinyImage.
- **Audio.** It is a 192-dimensional dastset that is composed of about 0.05 million audio feature vectors from DARPA TIMIT audio speed dataset.
- **Mnist.** The Mnist dataset contains about 0.07 million images of hand-written digits, which are represented as 784-dimensional vectors.
- **Notre.** It has about 0.3 million 128-dimensional features of a set of Flickr images and a reconstruction.

- **Sift.** The Sift dataset contains 1 million 128-dimensional SIFT vectors.
- **Deep.** The Deep dataset has 1 million data points with 256 dimensions that are deep neural codes of natural images obtained from the activations of a convolutional neural network.
- **Ukbench.** It is about 1 million 128-dimensional features of images.
- **ImageNet.** The ImageNet consists of about 2.4 million data points with 150-dimensional dense SIFT features.
- **Sift10M.** This dataset contains 10 million 128-dimensional SIFT vectors.
- **Sift100M.** This dataset contains 100 million 128-dimensional SIFT vectors.
- **Deep10M.** This dataset contains 10 million 96-dimensional DEEP vectors.
- **Deep100M.** This dataset contains 100 million 96-dimensional DEEP vectors.

The parameters have an important influence on the performance of our method. To this end, we empirically determine some near optimal parameters with respect to different datasets. For data compression, each vector is divided into $M = \frac{d}{2}$ subvectors and the centroids of each subspace are $k^* = 256$. The details for building the pre-calculation table can refer to [10]. In addition, we select the number of hash function $m = 60$ and the collision threshold $l = 50$ as the default for the experiments.

5.2 Evaluation Metrics

We employ the following metrics to evaluate the performance of our algorithm.

- **Recall.** We employ recall as a criterion to measure the accuracy for different algorithms. For the k-NNS, the recall is defined as the fraction on how many the k points answered by an algorithm are appeared in the true k nearest neighbors. Hence, it can be formalized as

$$Recall = \frac{|R' \cap R|}{|R|}$$

  where R' is a set of k points answered for a query and R is a set of true k nearest neighbors for the query. In our experiment, the Recall is computed as $R1@1, R10@10, \ldots, R100@100$.
- **Query Answering Time.** Another evaluation metrics is the query answering time, which is defined as the wall-clock time of an algorithm to answer k-NN.

In our experiments, we report the average recall and average running time as the final results, where both of them are the average over the queries.

5.3 Baseline Algorithms

There are many the state-of-the-art algorithms for approximate nearest neighbor search (ANNS), such as QALSH [5], VHP [6], PMLSH [18], R2LSH [19], SRS [20], HD-index [21], PQBF [22]. Since R2LSH performs better for ANNS

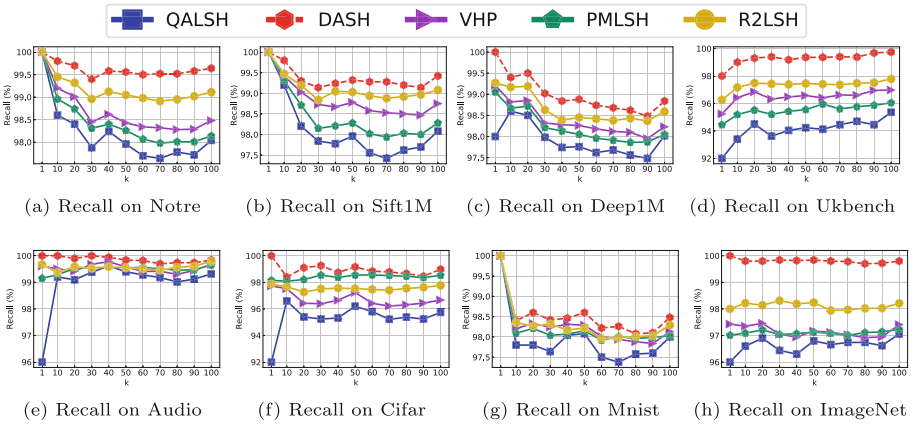
compared with SRS, Hd-index and PQBF [19]. Also, we find that VHP, PMLSH and R2LSH are not compared with each other. Hence we select QALSH, VHP, PMLSH, R2LSH as baselines. Note that those algorithms work on memory. To implement the best performance of VHP, we use some parameter values presented in [6], in which hash functions $m = 60$, success probability $P^* = 0.9$ and the initial search window $t_0 = 1.4$. For QALSH, we employ the improved version, which can achieve higher accuracy and support $c = 1$. VHP and QALSH use identical hash functions m and collision threshold l. For PMLSH, we choose the parameters proposed in [18], with $m = 15$, $P^* = 1 - 1/e$ and the number of pivots $s = 5$. Also, the parameters of R2LSH are set as the default value suggested by the authors in [19], where $m = 40$ and $P^* = 0.9$. For the c-k-ANNS, we set $c = 1$ and $k \in \{1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$.

5.4 Results and Analysis

Our method is based on the framework of QALSH. Due to the employment of PQ, our method requires to add extra consumption to construct indexing compared with QALSH, whereas the consumption is relatively small. For example, with respect to the large-scale dataset Deep100 M of size 36 GB used in the experiments, the additional time and space consumption are around 470 s and 4.5 GB, respectively, where QALSH needs about 46 GB memory space for constructing hash tables and the corresponding time is close to 820 s. It can be found that the additional time and memory space are only around a half and tenth of QALSH, respectively.

General-Scale Data. We study the performance mainly focusing on the average recall and query answering time. The distance for any two points is estimated with PQ, it is inevitable to result in certain estimated error and the destruction of probability guarantee. Nevertheless, our method can speed up the search processing with the pre-calculation distance table and obtain the similar probability guarantee with LSH based on residual prior distribution. Since the distance calculation with PQ negatively impacts the search accuracy, we use the original data to solve this issue. More explicitly, we consider that the real k nearest neighbors for any query are within the top-k' points returned, where $k' > k$ is a predefined constant. If we reorder the top-k' points with Euclidean distance, then it has higher possibility to find top-k exact nearest neighbors. Hence, we answer the best top-k in top-k' points as the final results for any query to achieve higher accuracy. The average recall $Rk@k$ by varying k from 1 to 100 under the success probability $P^* = 0.9$ is given in Fig. 3. One can be found that the average recall $Rk@k$ for our method is almost higher than other state-of-the-art methods with respect to different datasets.

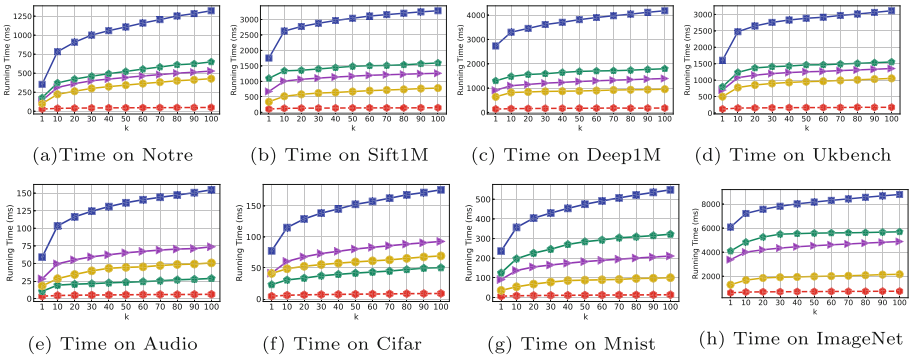
Correspondingly, the running time curves for k-NNS are shown in Fig. 4. One can observe that the running time for R2LSH, PMLSH and VHP presents different over various datasets, while DASH is lower significantly than them. This is because when the candidates have been retrieved, other methods need to calculate the Euclidean distance, which cost a large amount of time; by contrast,



(a) Recall on Notre (b) Recall on Sift1M (c) Recall on Deep1M (d) Recall on Ukbench

(e) Recall on Audio (f) Recall on Cifar (g) Recall on Mnist (h) Recall on ImageNet

Fig. 3. The comparison on the accuracy among different methods.



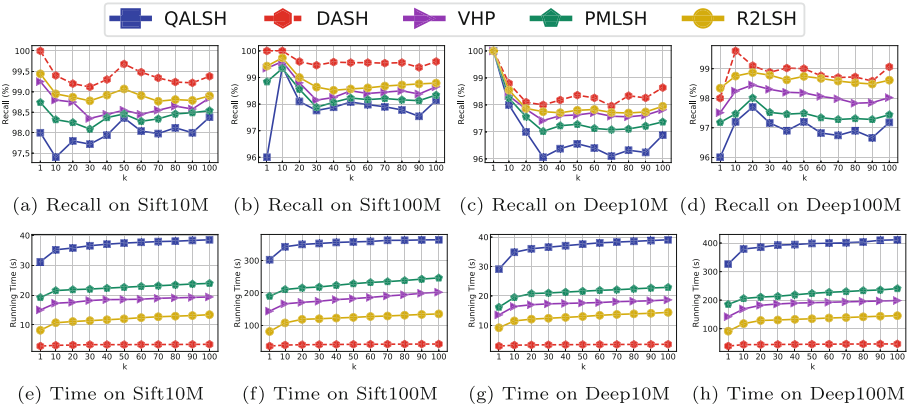
(a)Time on Notre (b) Time on Sift1M (c) Time on Deep1M (d) Time on Ukbench

(e) Time on Audio (f) Time on Cifar (g) Time on Mnist (h) Time on ImageNet

Fig. 4. The comparison on the running time among different methods.

DASH only needs to calculate the approximate distance by the pre-calculation distance table, so that the time consumption is relatively less. This means that DASH is more promising as k varies. For the dataset Mnist with high dimensionality, DASH performs better than those with low dimensionality, in which the speed can reach up to more than 40x in comparison with QALSH. Generally, DASH can achieve at least 5x speedup than other methods. Therefore, DASH is more superior pertaining to high dimensional datasets. Note that the cost of running time for finding k nearest neighbors is proportional to the increment of k, while the corresponding curve for DASH looks pretty stable than other algorithms because the magnitude of increment is relatively small.

Large-Scale Data. When DASH is applied to process more large-scale data, it also has significant superiority on accuracy and running time, as shown in Fig. 5. From the results, we can see that the accuracy obtain by DASH is higher than other algorithms with k increasing. The main reason is that DASH could access



(a) Recall on Sift10M (b) Recall on Sift100M (c) Recall on Deep10M (d) Recall on Deep100M

(e) Time on Sift10M (f) Time on Sift100M (g) Time on Deep10M (h) Time on Deep100M

Fig. 5. The comparison on the search performance among different methods.

more points to obtain desired accuracy under the suitable success probability. In addition, DASH is able to achieve at least 4x speedups than other algorithms. It is benefited from the acceleration property of PQ. This indicates that the search performance of DASH has prominent superiority than other algorithms as the scale of datasets increases.

6 Related Work

Approximate nearest neighbor search (ANNS) has attracted extensive attention over decades. There exists a vast majority of works to solve the ANNS problem. For example, the space partitioning methods [12,13] perform well in the low-dimensional space, while their performance greatly decreases due to the"curse of dimensionality". The quantization-based methods play an important role in data compression at the cost of bringing the quantization error, e.g. PQ [10], such that query accuracy is relatively lower. Hence, many methods have been proposed to decrease the quantization error, such as OPQ [14] and TQ [15]. The graph-based methods [16,17] have favourable results for high-dimensional ANNS, which are benefited from effective indexing structure. Although it could reach up to high recall with few time, they are lack of quality guarantee. In addition, the hash-based methods employ a family of hash functions mapping the nearby points to the same bucket with high probability than the distant points. However, LSH needs to construct many hash tables to achieve desired accuracy. For this drawback, many LSH-based variants have been proposed, e.g., [18,19].

7 Conclusion

In this paper, we propose a time efficient data-dependent hashing sheme called Data Aware Sensitive Hashing (DASH) for approximate nearest neighbor search

in high-dimensional space. DASH is based on the search framework of QALSH and takes the residual distance prior into account to evaluate a common distribution family for achieving probability guarantee. The extensive experiments are conducted to verify the efficiency and effectiveness of DASH by employing several real-life datasets. The results show that DASH obtains better search performance under the same reported quality compared against other methods.

Acknowledgements. The work reported in this paper is partially supported by NSF of Shanghai under grant number 22ZR1402000, the Fundamental Research Funds for the Central Universities under grant number 2232021A-08, State Key Laboratory of Computer Architecture (ICT,CAS) under Grant No. CARCHB 202118, Information Development Project of Shanghai Economic and Information Commission (202002009).

References

1. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of ACM STOC, pp. 604–613 (1998)
2. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.: Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of SoCG, pp. 253–262 (2004)
3. Ren, Z., Gu, Yu., Li, C., Li, F.F., Yu, G.: GPU-based dynamic hyperspace hash with full concurrency. Data Sci. Eng. **6**(3), 265–279 (2021). https://doi.org/10.1007/s41019-021-00161-5
4. Gan, J., Feng, J., Fang, Q., Ng, W.: Locality-sensitive hashing scheme based on dynamic collision counting. In: Proceedings of SIGMOD, pp. 541–552 (2012)
5. Huang, Q., Feng, J., Zhang, Y., et al.: Query-aware locality-sensitive hashing for approximate nearest neighbor search. In: Proceedings of VLDB, pp. 1–12 (2015)
6. Lu, K., Wang, H., Wang, W., Kudo, M.: VHP: approximate nearest neighbor search via virtual hypersphere partitioning. In: Proceedings of VLDB, pp. 1443–1455 (2020)
7. Andoni, A., Razenshteyn, I.: Optimal data-dependent hashing for approximate near neighbors. In: Proceedings of STOC, pp. 793–801 (2015)
8. Andoni, A., Naor, A., Nikolov, A., et al.: Data-dependent hashing via nonlinear spectral gaps. In: Proceedings of ACM SOTC, pp. 787–800 (2018)
9. Gao, J., Jagadish, H.V., et al.: DSH: data sensitive hashing for high-dimensional k-nnsearch. In: Proceedings of SIGMOD, pp. 1127–1138 (2014)
10. Jegou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. IEEE Trans. Pattern Anal. Mach. Intell. **33**(1), 117–128 (2010)
11. Dong, W., Wang, Z., Josephson, W., et al.: Modeling lsh for performance tuning. In: Proceedings of CIMK, pp. 669–678 (2008)
12. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: Proceedings of SIGMOD, pp. 47–57 (1984)
13. Bustos, B. Pedreira, O. Brisaboa, N.: A dynamic pivot selection technique for similarity search. In: Proceedings of SISAP, pp. 394–401 (2008)
14. Ge, T., He, K., Ke, Q., Sun, J.: Optimized product quantization for approximate nearest neighbor search. In: Proceedings of CVPR, pp. 2946–2953 (2013)
15. Babenko, A., Lempitsky, V.: Tree quantization for large-scale similarity search and classification. In: Proceedings of CVPR, pp. 4240–4248 (2015)

16. Yi, P., Li, J., Choi, B., Bhowmick, S.S., Xu, J.: FLAG: towards graph query autocompletion for large graphs. Data Sci. Eng. **7**(2), 175–191 (2022)
17. Malkov, Y.A., Yashunin, D.A.: Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE Trans. Pattern Anal. Mach. Intell. (2018)
18. Zheng, B., Xi, Z., Weng, L. et al.: PM-LSH: A fast and accurate LSH framework for high-dimensional approximate NN search. In: Proceedings of VLDB, pp. 643–655 (2020)
19. Lu, K. and Kudo, M.: R2LSH: A nearest neighbor search scheme based on two-dimensional projected spaces. In: Proceedings of ICDE, pp. 1045–1056 (2020)
20. Sun, Y., Wang, W., Qin, J., et al.: SRS: solving c-approximate nearest neighbor queries in high dimensional euclidean space with a tiny index. In: Proceedings of VLDB, pp. 1–12 (2014)
21. Arora, A., Sinha, S., Kumar, P., Bhattacharya, A.: Hd-index: Pushing the scalability-accuracy boundary for approximate knn search in highdimensional spaces. In: Proceedings of VLDB, pp. 906–919 (2018)
22. Liu, Y, Cheng, H, Cui, J.: PQBF: I/O-efficient approximate nearest neighbor search by product quantization. In: CIKM, pp. 667–676 (2017)
23. Satuluri, V., Parthasarathy, S.: Bayesian locality sensitive hashing for fast similarity search. In: Proceedings of VLDB, pp. 430–441 (2012)