Research Article

A Novel Accuracy and Similarity Search Structure Based on Parallel Bloom Filters

Chunyan Shuai,¹ Hengcheng Yang,¹ Xin Ouyang,² Siqi Li,¹ and Zheng Chen³

¹Faculty of Electric Power Engineering, Kunming University of Science and Technology, Kunming 650051, China

²Faculty of Information Engineering and Automation, Kunming University of Science and Technology, Kunming 650051, China

³Faculty of Transportation Engineering, Kunming University of Science and Technology, Kunming 650051, China

Correspondence should be addressed to Xin Ouyang; kmoyx@qq.com

Received 21 April 2016; Revised 25 September 2016; Accepted 26 October 2016

Academic Editor: Hong Man

Copyright © 2016 Chunyan Shuai et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

In high-dimensional spaces, accuracy and similarity search by low computing and storage costs are always difficult research topics, and there is a balance between efficiency and accuracy. In this paper, we propose a new structure Similar-PBF-PHT to represent items of a set with high dimensions and retrieve accurate and similar items. The Similar-PBF-PHT contains three parts: parallel bloom filters (PBFs), parallel hash tables (PHTs), and a bitmatrix. Experiments show that the Similar-PBF-PHT is effective in membership query and K -nearest neighbors (K -NN) search. With accurate querying, the Similar-PBF-PHT owns low hit false positive probability (FPP) and acceptable memory costs. With K -NN querying, the average overall ratio and rank- i ratio of the Hamming distance are accurate and ratios of the Euclidean distance are acceptable. It takes CPU time not I/O times to retrieve accurate and similar items and can deal with different data formats not only numerical values.

1. Introduction

In high-dimensional spaces, exact search methods, such as kd -tree approaches and Q -gram, are only suitable for small size vectors due to huge computation resources. However, similar search algorithms can drastically improve the search speed while maintaining good precision [1], which include VA-files, best-bin-first, space filling curves, K -means (see [2] and references therein), NV tree [3], K -nearest neighbors (K -NN), and locality-sensitive hashing (LSH) [4]. Most K -NN methods adopt the Euclidean distance; they assume all coordinates are numerical and own same units and semantics. But, in some applications, the dimension may be string or category, which makes the Euclidean distance questionable and artificial.

In query tools, a bloom filter [5] (BF), as a space-efficient and constant query delay random data structure, has been applied to present a big set and retrieve memberships broadly [6]. But the BF only can present 1-dimensional elements of a set; references [7–9] extended it to present high-dimensional sets and dynamic sets. But these methods can only answer the membership query, not the similarity query. In [10, 11], the

LSH functions replace the random hash functions of the BF to implement the similarity search, while [10, 11] only can deal with numerical coordinates and return the elements whose distances from the query are at most CR distance in Euclidean spaces, which lead to false negative probability (FNP).

Here, by computing the Hamming distance, we propose a new structure, called Similar-PBF-PHT, based on the BFs and hash tables (HT) to search the membership as well as the K -NN regardless of the radius CR. The Similar-PBF-PHT includes PBFs, PHTs, and a bitmatrix. The PBFs and PHTs apply d BFs and HTs to store d dimensions, and the bitmatrix stores the dependences of the dimensions. The experiments show that the Similar-PBF-PHT owns better performance in Hamming spaces than other methods. Meanwhile, with K -NN searching, it gets a balance performance and can process different data formats while other LSH-based methods can only deal with numerical value.

2. Related Work

There are different kinds of approximate search algorithms, and we divide them into three categories to discuss.

The famous one is space partition method, including IDistance [12] and MedRank [13]. The IDistance [12] clusters all high-dimensional elements into multiply spaces and converts them into 1-dimension space. It costs linear space and supports data insertion and deletion; however, if the data distribute uniformly or dimensions are anisotropic, space partition and center selection will be difficult. The MedRank [13] is a rank aggregation and instance optional algorithm, which aggregates the given dataset into M sorted lists, where every element has an entry with a form (Id, key). The number of the lists M equals $\log n$, where n is the number of the elements, and by M lists probing, the MedRank finds out approximate NN items. The MedRank possesses the best linear-space, but an element insertion or deletion needs update M lists and every list requires sorting again.

The LSH and its variants are other famous K -NN search algorithms [14], like Rigorous-LSH [15], E2LSH [4, 16], Adhoc-LSH [17], LSB-tree [18], LSB-forest [18], BLSH [11], and so on [19–21]. Let S be a set of points in d -dimensional space, The Rigorous-LSH [15] applies C-approximate ball cover, which has radius R and centers at the query point q , denoted as $B(q, R)$. If $B(q, CR)$ contains at least one point in S ($C \geq 1$ is a constant), it returns a point that is at most CR distance to q ; others return nothing. The Rigorous-LSH is theoretically perfect, but the query and space costs are expensive. In Euclidean spaces, E2LSH [4, 16] achieves the Rigorous-LSH through p -stable distribution [22], which reduces CPU and memory costs of the Rigorous-LSH greatly. The Adhoc-LSH [17] modifies the drawbacks of the Rigorous-LSH by a heuristic approach. Let a query be q and a magic radius RM ; the Adhoc-LSH returns the points within the radius of RM . If the RM equals the distance between the q and the exact NN, the Adhoc-LSH works well. If not, an improper RM may lead to FNP. Beyond locality sensitive hashing (BLSH) [11] scheme uses a two-level hashing algorithm to overcome the lower bound of the FNP [19] and finds CR-NN in Euclidean spaces. Different from other LSH methods, the second-level BLSH, parameterized by different center points, is a data-aware scheme. The outer hash table partitions the data sets into buckets of bounded diameter. For each bucket, the BLSH constructs an inner hash table, which applies the minimum enclosing ball of the points in the bucket as a center point. However, the BLSH still has high memory costs and will bring FNP. The LSB-tree and LSB-forest [18] implement the K -NN search by space mapping and Z-order coding. By the LSH functions, the LSB-tree [18] first maps d dimensional points $S(o)$ to a lower dimensional points $G(o)$. Then the LSB-tree gets the Z-order [23] value of the $G(o)$, which is indexed by a conventional B-tree. Multiply LSB-trees form a LSB-forest, which can update efficiently and satisfy query accuracy but space costs are expensive.

The BFs are introduced into the high-dimensional search, like high-dimensional dynamic BFs (MDDDBFs) [7], PBF-BF [8], PBF-HT [8], similarity sets [24], and distance-sensitive BFs (DSBF) [10], and so on [25]. The MDDDBFs [7] apply d parallel standard BFs (PBFs) to present a d -dimensional dynamic dataset. By searching d PBFs, the MDDDBFs find out the membership, but the MDDDBFs lack a way to verify the dependency of multiple dimensions of an item, which

causes high FPPs with membership retrieval. To reduce the FPP, PBF-BF and PBF-HT [8] add another BF and hash table (HT) to the PBFs to store the verification value of the different dimensions. However the methods above based on the BFs can only answer the membership query, not similarity query.

Distance-sensitive BFs (DSBF) [10] replace the uniform hash functions in the BF with the LSH functions to find out similar strings. But the DSBF can only differentiate a query string that differs from all strings in the dataset on a (constant) δ -fraction of bits. The locality-sensitive bloom filter (LSBF) [26] uses two-level BFs to implement the approximate item query. The first-level bloom replaces the random hash functions with locality-sensitive hash function (LSH), which is based on p -stable distribution [22], and maps all items to m bit-bloom arrays. To keep the integrity and reduce the FPP, the second level BF stores the hash verification signature formed by the k LSH functions in the first-level BF. In order to reduce the FNP, the LSBF needs to probe the neighbor bits in the first-level BF, which leads to cost more query time. Meanwhile since the LSH function concentrates most points around the mean and maps some neighboring points to remote bits, it will bring bigger FPP and FNP.

3. Structures and Working Mechanism

3.1. Structures. A standard BF [5] applies an array of m bits (initially all are set to 0) and k independent hash functions h_i to represent a set $s = \{a_1, a_2, \dots, a_n\}$ of n elements, as shown in Figure 1(a). If an element is mapped into the BF by h_i , the corresponding bit $h_i(a_i) \% m$ is set to 1. Given a query q , by k hash functions $h_i(q) \% m$ mapping, the BF answers whether the q is a member of S with a FPP. In order to support elements deletion, counting bloom filter (CBF) [27, 28] replaces the array of m bits with m counters.

In this paper, to present high-dimensional elements, d parallel BFs (PBFs) and parallel hash tables (PHTs) are proposed to represent the elements with d dimensions. At the same time, a bitmatrix is introduced to keep the inherent dependency of the dimensions and reduce the FPP, as shown in Figure 1(b).

3.1.1. PBFs. To store the dimensions, this paper introduced d BFs (Figure 1(b)), and every BF owns k independent hash functions [29] $h_1, h_2, \dots, h_k$ and a bit array with m length, denoted as B . Let a_j present the j th dimension and $h[j][i](a_j)$ present i th hash function value of the j th dimension. When a_j is mapped into B_j , the corresponding place $B_j[h[j][i](a_j) \% m]$ is set to 1. Since attenuation method can make hash values distribute broadly [8] and reduce the FPP, we apply the attenuation sum of k hash values $F(1) = \sum h[j][i](a_j)/2^i$ to store the verification value of the dimension a_j .

3.1.2. PHTs. In order to find out which dimensions and how many dimensions of the elements in set S are similar to the query q , this paper utilizes d parallel hash tables (PHTs) and hash links to store identifications (IDs) of the elements. Each hash table, denoted as HT, is indeed a link array with m_2 length.

FIGURE 1: Bloom filter and Similar-PBF-PHT.

3.1.3. Bitmatrix. Since d dimensions are stored into d BFs and HTs separately, the integrity of the elements is destroyed, which leads to query confusion. Thus, an auxiliary structure, called bitmatrix, is added to record dimensions hit in the PBFs and PHTs. After d dimensions are checked in the PBFs and PHTs, numbers of the hit dimensions are summed up in the bitmatrix; that is, $F(2) = \max(\sum_{i=1}^d d_i)$. If $F(2) = d$, the query is a member of the set S with a FPP, as shown in (1). If $F(2) = 0$, no dimension of the query is in the set, for example, (2). If $0 < F(2) < d$ the query is a similar elements with a FPP, as shown in (3).

The Bitmatrix

(a) Membership:

$$\begin{array}{c}
 \begin{array}{c} n_1 \quad n_2 \quad n_3 \quad n_4 \quad n_5 \quad n_6 \quad n_7 \quad n_8 \\
 d_1 \quad \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \\
 d_2 \quad \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \\
 d_3 \quad \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \\
 d_4 \quad \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \\
 d_5 \quad \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \\
 \hline
 \sum d_i \quad 3 \quad 3 \quad 1 \quad 2 \quad 5 \quad 1 \quad 0 \quad 0
 \end{array}
 \end{array} \quad (1)$$

(b) Outlier:

$$\begin{array}{c}
 \begin{array}{c} n_1 \quad n_2 \quad n_3 \quad n_4 \quad n_5 \quad n_6 \quad n_7 \quad n_8 \\
 d_1 \quad \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\
 d_2 \quad \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\
 d_3 \quad \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\
 d_4 \quad \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\
 d_5 \quad \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\
 \hline
 \sum d_i \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0
 \end{array}
 \end{array} \quad (2)$$

(c) Similarity:

$$\begin{array}{c}
 \begin{array}{c} n_1 \quad n_2 \quad n_3 \quad n_4 \quad n_5 \quad n_6 \quad n_7 \quad n_8 \\
 d_1 \quad \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \\
 d_2 \quad \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\
 d_3 \quad \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \\
 d_4 \quad \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \\
 d_5 \quad \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \\
 \hline
 \sum d_i \quad 3 \quad 3 \quad 1 \quad 2 \quad 4 \quad 1 \quad 0 \quad 0
 \end{array}
 \end{array} \quad (3)$$

3.2. Working Mechanism

3.2.1. Initialization or Insertion. When d dimensions of an element are mapped into d BFs and HTs by k hash functions, the locations of the bit array in the PBFs are set to 1, and the attenuation hash values are summed up: $F(1) = \sum_{i=1}^k h[j][i](a_j)/2^i$. By $F(1)\%m_2$ mapping, the corresponding link in the j th HT is found, and a new hash node is added to the tail of the link to store the item's ID.

3.2.2. Query. Only when the dimension returns 1 in the BF will the attenuated hash values be summed up and located in the corresponding HT. The hit elements' IDs are found and the corresponding bits in the bitmatrix are set to 1. After all d dimensions are mapped, columns in the bitmatrix are summed up. If the summation is in the range between 1 and d , the membership or similar elements are obtained.

3.2.3. Element Deletion. Since bit deletion in a BF will bring FPP; the Similar-PBF-PHT only needs to delete the hash node in the corresponding HT.

4. Performance Analysis

Since the BF only has FPP but not FNP [24], we evaluate the performance of the Similar-PBF-PHT by the quality of results, FPP, query time, and space consumption.

4.1. False Positive Probability (FPP)

False Positive. A query q is a false positive to object set S , if the query gets a positive answer while in fact q is not a membership, or there does not exist a neighbor object p in Euclidean or Hamming spaces. FPP is the probability of false positives.

4.1.1. FPP of a BF and a HT

Theorem 1. Where n elements in the set S have been mapped to a BF with m bits by k different independent hash functions, the FPP of a BF [24] is

$$\begin{aligned} FPP_{BF} &= (1 - p')^k = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \\ &\approx (1 - e^{-kn/m})^k. \end{aligned} \quad (4)$$

When $k = (m/n) \ln 2$, the f_{BF} obtains the minimum value $(1/2)^k$ or $0.6185^{m/n}$ [24].

Theorem 2. A HT possess $f_{HT_{max}} = 2\phi((\sqrt{3} * 2^k n)/2(q - 1)\sqrt{k}) - 1$ upper bound FPP, if k hashes of the BF follow uniform distribution and the attenuated check value $v(a_j) = \sum_{i=1}^k h_i(a_j)/2^i$ obeys normal distribution.

Proof. In this paper, d BFs are used to present the d dimensions of elements, and each BF owns k independent hash functions [19]. Let each random variable $h_i(a_j)$ follow the uniform distribution with range $\{1, \dots, M\}$, the expected value of $(1 + M)/2$, and variance of $(M - 1)^2/12$, and $v(a_j) = \sum_{i=1}^k h_i(a_j)$ is ranged $\{k, \dots, kM\}$. According to central limit theorem [1], if k is big enough, the random variable $v(a_j)$ satisfies a normal distribution with the expected value of $k(1 + M)/2$ and variance of $k(M - 1)^2/12$. Because the sum of attenuated value of k hash functions $v(a_j) = \sum_{i=1}^k h_i(a_j)/2^i$ can reduce the FPP, we store the attenuated value of j th dimension of element a into the j th HT. It is difficult to estimate the probability density functions of $v(a_j)$, and according to birthday attack [2], when $v(a_j)$ distributes uniformly, the collision will be minimum. For simplicity, we suppose $v(a_j)$ satisfies normal distribution to estimate the upper bound of false positive.

Let $v'(a_j) = \sum_{i=1}^k h_i(a_j)/2^k$; due to $\sum_{i=1}^k h_i(a_j)/2^i > \sum_{i=1}^k h_i(a_j)/2^k$, the discrete degree of $v(a_j)$ is bigger than $v'(a_j)$, while the collision probability of $v(a_j)$ is smaller than $v'(a_j)$. Let $p(v(x_j))$ be the FPP of the verification value of the j th attribute of x . There exists $p(v(x_j)) < p(v'(x_j))$. According to central limit theorem [30], $v'(a_j) = \sum_{i=1}^k h_i(a_j)/2^k$ satisfy

normal distribution, with the expected value of $\mu = k(1 + M)/(2 * 2^k)$ and variance of $\delta^2 = k(M - 1)^2/(12 * 2^k)$.

Let there be n items and $v'(a_j)$ be in the range of $\{1, \dots, M_1\}$. In order to get the upper bound, we compute the maximum FPP of each dimension, that is, $f_{HT_{max}}$:

$$\begin{aligned} f_{HT_{max}} &= p(v(a_j)) \leq p(v'(a_j)) \\ &= p\left(\mu - \frac{n}{2} \leq v'(a_j) \leq \mu + \frac{n}{2}\right) = 2\phi\left(\frac{n}{2\delta}\right) - 1 \\ &= 2\phi\left(\frac{\sqrt{3} * 2^k n}{2(q - 1)\sqrt{k}}\right) - 1. \end{aligned} \quad (5)$$

□

4.1.2. FPP of Similar-PBF-PHT

Theorem 3. With K -NN searching, the average FPP of i dimensions misdetected, $f_{similar-PBF-PHT}^i$ is $((1 + f_{BF-HT})^i - 1)/(2^i - 1)$, in which f_{BF-HT} is the FPP of a dimension misdetected in both the BF and HT. When $i = d$, it is membership search, and the hit FPP $f_{similar-PBF-PHT}^{hit}$ is $((1 + f_{BF-HT})^d - 1)/(2^d - 1)$.

Proof. The BF and HT are independent, so the FPP of a dimension is $f_{BF-HT} = f_{BF} * f_{HT}$. With similarity querying, if any i dimensions collide in the BFs and HTs and other $i - 1$ dimensions are members, the collision happens, which satisfies binomial distribution, and all combinations are

$$\begin{aligned} T_c &= c_d^{i-1} c_{d-i+1}^1 + c_d^{i-2} c_{d-i+2}^2 + \dots + c_d^0 c_d^i \\ &= \frac{d!}{(d-i)!} \sum_{j=0}^i \frac{1}{(k-j)! j!} - c_d^i c_{d-i}^0 = c_d^i (2^i - 1). \end{aligned} \quad (6)$$

With similarity searching, let $f = f_{BF-HT}$, and the average FPP $f_{similar-PBF-PHT}^i$ is

$$f_{similar-PBF-PHT}^i = \frac{1}{T_c} (c_d^{i-1} c_{d-i+1}^1 f^1 + \dots + c_d^0 c_d^i f^i). \quad (7)$$

When $0 \leq f \leq 1$, there is $C_i^0 f^0 + C_i^1 f^1 + \dots + C_i^i f^i = (1 + f)^i$, and the FPP is

$$\begin{aligned} f_{similar-PBF-PHT}^i &= \frac{1}{T_c} (c_d^{i-1} c_{d-i+1}^1 f^1 + \dots + c_d^0 c_d^i f^i) \\ &= \frac{1}{T_c} (c_d^i (c_i^1 f^1 + c_i^2 f^2 + \dots + c_i^i f^i)) \\ &= \frac{(1 + f)^i - 1}{2^i - 1}. \end{aligned} \quad (8)$$

When $i = d$, it is the membership search, namely, $f_{similar-PBF-PHT}^{hit}$. □

If d dimensions are misdetected simultaneously, f_{BF-HT}^d gets the minimum value. If only one dimension is falsely detected, $f_{similar-PBF-PHT}^{hit}$ is the maximum value f_{BF-HT} . There is

$$f_{BF-HT}^d \leq f_{similar-PBF-PHT}^{hit} \leq f_{BF-HT}. \quad (9)$$

FIGURE 2: The maximum and minimum value of $\log f^{\text{hit}}$.

In Figure 2, let $f_{\text{BF-HT}} = 0.5$; with dimensions increasing, the maximum and minimum of the $f_{\text{similar-PBF-PHT}}^{\text{hit}}$ decay exponentially.

4.2. Average Overall Ratio. We evaluate the quality of a K -NN search result by rank- i ratio and average overall ratio (AOR) [18], which are used in most experiments. The rank- i ratio is denoted by $R_i(q)$ and defined as

$$R_i(q) = \frac{\|o_i, q\|}{\|o_i^*, q\|}, \quad (10)$$

where $i \in [1, K]$, $\|o_i, q\|$ is the distance of the queried i th neighbor to q , and $\|o_i^*, q\|$ is the distance of the actual i th neighbor to q . The overall approximation ratio is the mean of the ratios of all ranks, namely, $(\sum_{i=1}^K R_i(q))/K$.

4.3. Complexity

4.3.1. Storage Space. The storage spaces of the Similar-PBF-PHT contain three parts:

(i) When the FPP of a BF is not greater than ε and the number of hash functions is optimal, to express the set S of n elements, the size of the BF array must be $m \geq n(\log_2(1/\varepsilon)/\ln 2) = -n\log_2 e \times \log_2 \varepsilon$. Then the spaces required by d parallel BFs are (bits)

$$m \geq dn \frac{\log_2(1/\varepsilon)}{\ln 2} = (-\log_2 e \log_2 \varepsilon) dn. \quad (11)$$

(ii) A HT needs to store all IDs of the elements and the next node. Let αn be the length of the HT ($0 < \alpha$), a node takes up l_1 bits, and the HT requires spaces with a range from nl_1 to $(\alpha*n+n)l_1$. The spaces range of the d HTs is $[ndl_1, nd(1+\alpha)l_1]$ bits.

(iii) The bitmatrix needs nd bits.

The storage spaces of the Similar-PBF-PHT are (bytes)

$$\begin{aligned} \text{Space} &= \frac{(-nd \log_2 e \log_2(\varepsilon) + nd(1+\alpha)l_1 + nd)}{8} \\ &= \frac{nd}{8} (-\log_2 e \log_2 \varepsilon + (1+\alpha)l_1 + 1). \end{aligned} \quad (12)$$

When $\alpha = 0$, space gets the minimum value. and when l_1, ε , and α are constants, the space complexity of the Similar-PBF-PHT is

$$O(c_1 nd). \quad (13)$$

4.3.2. Search Time. When querying, the PBFs need kd times hash calculation, the PHTs require dl_2 times to search the IDs, in which l_2 is the average length of the HT bucket links.

During the membership searching, all hit element's IDs need to be recorded, and the time complexity is

$$O(dk + dl_2). \quad (14)$$

The bitmatrix traverses at most K times. So the time complexity is

$$O(dk + dl_2 + K). \quad (15)$$

5. Experiments

5.1. Dataset and Setting. The BF is designed to represent a set, and there is no benchmark. Here we choose 4 datasets used in most experiments; they are Color [13], Mnist [12], Varden [18], and Reuters 21578 [31]. Data formats in the Reuters 21578 are various including digital, character, symbols, and their combinations. We use it to generate 49396-item dataset with 1000 dimensions to test the performance of the Similar-PBF-PHT, including the query latency and the ability of data processing. The experiments run on a computer with 2.5 GHz Intel double Core processors and 8 G RAM.

5.2. Membership Query. In this section, we will discuss performances of different methods in membership query.

Let $d = 3$, $m = 320(640, 1280, 2560)$, $\alpha = 1.1$, and $l_v = 32$, where l_v is bits of every verification values in the PBF-HT and Similar-PBF-PHT. Figure 3 displays the FPPs of the SBF [5], PBF [7], PBF-HT [8], PBF-BF [8], and Similar-PBF-PHT on Reuters 21578 data. With the m increasing, the FPPs decrease, and to a constant m , the FPPs will increase with the number of the elements growing, especially the SBF and PBF.

FIGURE 3: FPPs of the SBF, PBF, PBF-HT, PBF-BF, and the Similar-PBF-PHT under 3 dimensions. (a) Address space $m = 320$. (b) Address space $m = 640$. (c) Address space $m = 1280$. (d) Address space $m = 2560$.

When the number of the items exceeds a threshold ($n \geq m$), the FPPs of the SBF and PBF are nearly equal to 1, which is consistent in the BF theory. In different m , the Similar-PBF-PHT gets the lowest FPP; even when $m = 320$, the biggest FPP is not beyond 0.01, while the FPPs of others are almost 1.

Figure 4 demonstrates memory usages of the PBF, PBF-BF, PBF-HT, and Similar-PBF-PHT on the Reuters 21578 dataset, when $FPP = 0.00098$, $d = 4$, $k = 5$, $n = [3000, 30000]$, and $l_v = 32$. According to formula (13), to fit a constant FPP, the memory usage will grow with the number of the items increasing. The hash tables and the bitmatrix reduce the FPP at the cost of memory, and the BF's bits arrays take up 1/4 spaces of the CBFs in other 3 schemes. All these make the space consumption of the Similar-PBF-PHT just a little higher than the PBF but lower than the PBF-HT.

5.3. K -NN Search. To evaluate the accuracy of the K -NN search, we compare the average overall ratios of the Rigorous-LSH [15], MedRank [13], Adhoc-LSH [17], LSB-tree, and LSB-forest [18] with the Similar-PBF-PHT on the Color and Mnist dataset, as shown in Figure 5. Workload is set to 50, and 1–100 nearest neighbors are searched. In Hamming spaces, the ratios of the Similar-PBF-PHT are almost equal to 1. In Euclidean spaces, the ratios of the Similar-PBF-PHT are not stable; the overall ratios on the Mnist are almost as good as the LSB-forest, but the ratios on Color are a little higher and increase with the number of nearest neighbors. The main reasons are that the dimensions of the Mnist are sparse (most values are 0), and most Hamming distances are 0. While the dimensions of the Color are dense, a small distance (0.0001) in Euclidean spaces will be recognized as 1 in Hamming

FIGURE 4: Memory usages of the PBF, PBF-BF, PBF-HT, and the Similar-PBF-PHT.

spaces. All these make the accuracy decrease, but the ratios are still beyond 0.98.

Figure 6 displays average rank- i ratios of the Euclidean and Hamming distance. $\|o_i, q\|$ presents Hamming distance; because of the FPP of the BF, the actual distance is less than the query distance; there exists $R_i(q) \leq 1$. In Figure 6(a), with i increasing, the rank- i ratios of Hamming distance are stable and not lower than 0.985. On the Mnist (Figure 6(b)), rank- i ratios of Euclidean distances of the Similar-PBF-PHT are minimum, almost equal to 1, while, on the Color

(Figure 6(c)), the ratios of Euclidean distance increase slowly and are higher than the LSB-tree and LSB-forest's; when the rank $-i \geq 7$, it becomes lower than the MedRank.

Table 1 analyzes the memory consumption (MB) on the Varden dataset, setting $B = 4096$ bytes, $c = 2$, and $l = \sqrt{dn/B}$. Although the memory costs of the LSB-tree are less, its FPPs are higher than the LSB-forest, so we abandon it. The memory usages of the Similar-PBF-PHT are minimum while the Rigorous-LSH are maximum, and all consumption

FIGURE 5: Average overall ratios of [1-100]-NN on Color and Mnist.

increases with d and n . When $c = 2$ the memory costs of the LSB-Forest are almost as big as the Adhoc-LSH.

The Similar-PBF-PHT can deal the dimensions with different formats and lengths, and the length of dimension and number of samples will affect the query time. In Figure 7(a), we set n to 100, 500, and 1000, respectively, and every dimension contains 20 characters (big enough to most applications) to search 10-NN. With dimensions growing, the average query latency of the Similar-PBF-PHT increases linearly. Let

$n = 5000$, $K = 10$, and $K = 10$; Figures 7(b) and 7(c) demonstrate effects of different dimension's lengths on query delay with 10-NN searching. Average query latencies will increase with the numbers of the characters and dimensions. This is because most of the CPU time is wasted on processing the hash values of the characters.

In Figure 8, we analyze the effects of the parameters α and f_{BF} on the AORs and FPPs of the Similar-PBF-PHT. Let $n = 50000$, $k = 3$, $l_v = 32$, and $K \in [20-100]$ -NN and let test

FIGURE 6: Average rank- i ratios of [1-10]-NN on Color and Mnist.

workload be 10000. As shown in the Figure 8, under different α , even small α (0.01), and big f_{BF} (0.5), the Similar-PBF-PHT gets good query results and low FPPs. That means the PHT and the bitmatrix can effectively improve the detection accuracy. α affects the query accuracy much more than the FPP of the BF. With α increasing, the FPP decreases and the AOR increases; at the same time the space consumption increases.

6. Conclusions

In this paper, we propose a comprehensive structure, called Similar-PBF-PHT, to represent and search member and similar elements of a big dataset in high-dimensional spaces by computing Hamming distance. We analyze its working mechanism, FPP, and space and time complexity in detail. The experiments show that, with membership searching,

FIGURE 7: Average query latency of Similar-PBF-PHT when searching 10-NN. (a) Average query latency of Similar-PBF-PHT when samples n are 100, 500, and 1000. (b) Average similar query latency when $d \in [2-1000]$, $n = 5000$, and $k = 5$. (c) Average similar query latency when the length of the dimension is within the range $[40-400]$, $d = 50$, $k = 5$, and $n = 5000$.

compared with the PBF, PBF-HT, and PBF-BF, the Similar-PBF-PHT owns lower hit FPP by a low memory cost. The Similar-PBF-PHT costs less storage than the schemes based on the locality sensitive hash, including the Rigorous-LSH, LSB-forest, Adhoc-LSH, and BLSH. With K -NN items querying, it costs CPU time, not I/O times, which make it have less query latency. Meanwhile, the Similar-PBF-PHT computes hash values of all characters in each dimension, so it can deal with different data formats (chars, number, symbol,

and so on), and the number of characters will affect the query time. The average overall ratios (query accuracy) and the average rank- i ratios of the Hamming distance are accurate. All these advantages make it appropriate for representing and searching items in high-dimensional spaces, such as database and documents similar search.

Although the Similar-PBF-PHT can get good performance in Hamming spaces, memory costs and the FPP of Euclidean spaces for K -NN searching are still a little higher.

FIGURE 8: Average overall ratios and average FPP when searching [20–100]-NN on Color under different f_{BF} and α .

TABLE 1: Space consumption on Varden data (MB).

(a) Space versus cardinality n ($d = 50$)					
n	10000	25000	50000	75000	100000
Rigorous-LSH	895	3624	10323	18892	29016
Adhoc-LSH	57	231	660	1208	1855
LSB-forest	57	231	660	1208	1855
Similar-PBF-PHT	13	31	61	92	121

(b) Space versus dimensionality d ($n = 50000$)				
d	25	50	75	100
Rigorous-LSH	382	896	1563	2436
Adhoc-LSH	24	57	101	159
LSB-forest	24	57	101	159
Similar-PBF-PHT	31	61	93	120

In the future, we will study the local sensitive hash functions to replace the random hash functions and further reduce the storage spaces.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This work is supported by the National Nature Science Foundation of China (no. 61562056), Yunnan Province Education Department (nos. 2015Z055 and 2015Y073), and Yunnan Province Nature Science Foundation (nos. KKS201303125, KKS201304129, and KKS201404106).

References

- [1] H. Samet, *Foundations of Multidimensional and Metric Data Structures*, Morgan Kaufmann, Burlington, Mass, USA, 2006.
- [2] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–128, 2011.
- [3] H. Lejsek, F. H. Ásmundsson, B. P. Jónsson, and L. Amsaleg, "NV-tree: an efficient disk-based index for approximate search

- in very large high-dimensional collections,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 5, pp. 869–883, 2009.
- [4] A. Andoni and P. Indyk, “Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions,” *Communications of the ACM*, vol. 51, no. 1, pp. 117–122, 2008.
 - [5] H. Burton, “Space/time trade-offs in hash coding with allowable errors,” *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.
 - [6] S. Geravand and M. Ahmadi, “Bloom filter applications in network security: a state-of-the-art survey,” *Computer Networks*, vol. 57, no. 18, pp. 4047–4064, 2013.
 - [7] D. Guo, J. Wu, H. Chen, Y. Yuan, and X. Luo, “The dynamic bloom filters,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 1, pp. 120–133, 2010.
 - [8] B. Xiao and Y. Hua, “Using parallel bloom filters for multiattribute representation on network services,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 1, pp. 20–32, 2010.
 - [9] M.-Z. Xiao, Y.-F. Dai, and X.-M. Li, “Split bloom filter,” *Acta Electronica Sinica*, vol. 32, no. 2, pp. 241–245, 2004.
 - [10] Kirsch and M. Michael, “Distance-sensitive bloom filters,” in *Proceedings of the 8th Workshop on Algorithm Engineering and Experiments (ALENEX ’06)*, pp. 41–51, Philadelphia, Pa, USA, 2006.
 - [11] A. Andoni, P. Indyk, H. L. Nguyen, and I. Razenshteyn, “Beyond locality-sensitive hashing,” in *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1018–1028, SIAM, Portland, Ore, USA, January 2014.
 - [12] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang, “IDistance: an adaptive B+tree based indexing method for nearest neighbor search,” *ACM Transactions on Database Systems*, vol. 30, no. 2, pp. 364–397, 2005.
 - [13] R. Fagin, R. Kumar, and D. Sivakumar, “Efficient similarity search and classification via rank aggregation,” in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD ’03)*, pp. 301–312, San Diego, Calif, USA, June 2003.
 - [14] J. Wang, H. T. Shen, J. Song et al., “Hashing for similarity search: a survey,” <https://arxiv.org/abs/1408.2927>.
 - [15] P. Indyk and R. Motwani, “Approximate nearest neighbors: towards removing the curse of dimensionality,” in *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pp. 604–613, 1998.
 - [16] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, “Locality-sensitive hashing scheme based on p-stable distributions,” in *Proceedings of the 20th Annual Symposium on Computational Geometry (SCG ’04)*, pp. 253–262, ACM, June 2004.
 - [17] A. Gionis, P. Indyk, and R. Motwani, “Similarity search in high dimensions via hashing,” in *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB ’99)*, pp. 518–529, Edinburgh, UK, September 1999.
 - [18] Y. Tao, K. Yi, C. Sheng, and P. Kalnis, “Efficient and accurate nearest neighbor and closest pair search in high-dimensional space,” *ACM Transactions on Database Systems (TODS)*, vol. 35, no. 3, article 20, 2010.
 - [19] R. O’Donnell, Y. Wu, and Y. Zhou, “Optimal lower bounds for locality-sensitive hashing (except when q is tiny),” *ACM Transactions of Computation Theory*, vol. 6, no. 1, article 5, 2014.
 - [20] D. Gorisse, M. Cord, F. Precioso, and S. Philipp-Foliguet, “Fast approximate kernel based similarity search for image retrieval task,” in *Proceedings of the IEEE 19th International Conference on Pattern Recognition (ICPR ’08)*, pp. 1873–1876, Tampa, Fla, USA, December 2008.
 - [21] D. Gorisse, M. Cord, and F. Precioso, “Locality-sensitive hashing for chi2 distance,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 2, pp. 402–409, 2012.
 - [22] W. H. Lee, *Continuous and discrete properties of stochastic processes [Ph.D. thesis]*, The University of Nottingham, Nottingham, UK, 2010.
 - [23] G. M. Morton, “A computer oriented geodetic data base and a new technique in file sequencing,” Tech. Rep., IBM, Ottawa, Canada, 1966.
 - [24] A. Broder and M. Mitzenmacher, “Network applications of Bloom filters: a survey,” *Internet Mathematics*, vol. 1, no. 4, pp. 485–509, 2004.
 - [25] S. Tarkoma, E. R. Christian, and L. Eemil, “Theory and practice of bloom filters for distributed systems,” *IEEE Communications Surveys and Tutorials*, vol. 14, no. 1, pp. 131–155, 2012.
 - [26] Y. Hua, B. Xiao, B. Veeravalli, and D. Feng, “Locality-sensitive Bloom filter for approximate membership query,” *IEEE Transactions on Computers*, vol. 61, no. 6, pp. 817–830, 2012.
 - [27] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor, “Linear-time probabilistic counting algorithm for database applications,” *ACM Transactions on Database Systems*, vol. 15, no. 2, pp. 208–229, 1990.
 - [28] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, “Summary cache: a scalable wide-area Web cache sharing protocol,” *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, pp. 281–293, 2000.
 - [29] General Purpose Hash Function Algorithms, <http://partow.net/programming/hashfunctions/index.html>.
 - [30] A. R. Barron, “Entropy and the central limit theorem,” *The Annals of Probability*, vol. 14, no. 1, pp. 336–342, 1986.
 - [31] Reuters-21578 Text Categorization Collection, <http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html>.