Journal of Computers Vol. 31 No. 5, 2020, pp. 44-60
doi:10.3966/199115992020103105004

Orthogonal Range Search Approach Using FGBC-iDistance

Xinpan Yuan, Qingyun Liu, Songlin Wang, Zhigao Zeng*

School of Computer Science, Hunan University of Technology, Zhuzhou 412007, China
674712173@qq.com

Received 20 March 2019; Revised 20 July 2019; Accepted 28 August 2019

Abstract. Orthogonal range queries in high-dimensional data is extremely important and relevant. Not to modify the current index and use the inherent functionality of the existing indexing and retrieval mechanisms, there are three orthogonal range search approaches, including naïve, space and data-based approach. Naïve approach is to approximate orthogonal search by external query circle of iDistance. The space approach is mainly to break the orthogonal range search into multiple squares. A data-based approach by iDistance index is better than naïve and space. This paper proposed a more fine-grained partition on iDistance index, each part corresponded with a unique FGBC code (fine-grained bit code), which realizes the candidate sets filtered more precisely. The experimental results on the synthetic and real datasets proved that the FGBC-iDistance is correct and effective.

Keywords: FGBC-iDistance, high-dimensional, iDistance, orthogonal range

1 Introduction

Many emerging database applications such as image, time series, and scientific databases, manipulate high-dimensional data. In these applications, one of the most common but expensive operations is to look up objects in a high-dimensional database that are similar to a given search object. Since these data are high dimensional, this is a challenge for efficiently searching and mining data in a database. At the same time, for users interacting with data, two types of queries are often required: orthogonal range and *k*-nearest neighbor (*k*NN).

1.1 *k*-Nearest Neighbor (*k*NN) Search

*k*NN depends on the high-dimensional data index structures. All of these high-dimensional data index structures are designed to filter out high-dimensional data that is not related to user queries during *k*-nearest neighbor search, thereby reducing the space of index, reducing distance computations, and improving the efficiency of retrieval.

In the similarity measure of high-dimensional data and the characteristics of high-dimensional data, high-dimensional data indexes can be divided into vector space index structure and metric space index structure [1]. The vector space index structure mainly manages the data by using the relative position of data in the entire data space. Typical examples include the R-Tree [2] series, the KD-Tree [3] series, and the VA-File [4] series. The metric spatial index structure filters irrelevant data with the principle of triangular inequality. Typical algorithms are VP-Tree [5], M-Tree [6], IDistance [7], and so on.

1.2 The Orthogonal Range Search

Orthogonal range search has been applied in many fields of computer science, including databases and computer graphics, etc., so it has been widely studied [8-2]. Orthogonal range queries are used to solve text indexing problems [13]. Ishiyama and Sadakane published A succinct data structure for multidimensional **orthogonal range searching** in 2017 [14]. The Chan T M solved the orthogonal range

* Corresponding Author

searching problem and the exact nearest neighbor searching problem for a static set of n points when the dimension d is moderately large [15]. Chan and Tsakalidis Dynamic solved the best RAM upper bounds known to date for a number of problems, including 2-d orthogonal range emptiness, 3-d orthogonal range reporting, and offline 4-d dominance range reporting [16]. Bringmann, Husfeldt, and Magnusson show that the analysis in the regime of non-constant treewidth can be improved by revisiting the analysis of orthogonal range searching [17]. Given a collection of multidimensional data points and a query rectangle, the goal of an **orthogonal range search** is to retrieve all the data points that fall within a given rectangle. Apart from the applications of range search as such, it is implicitly involved in more complex region queries and other associative queries.

Orthogonal range search is the query of data points within a rectangular range whose edges are aligned with the coordinate axes (orthogonal) after preprocessing the set of points in the d-dimensional space. **Orthogonal range queries** differ in that they require a min and max value to be selected for every dimension of the search.

Not to modify the current index and use the inherent functionality of the existing indexing and retrieval mechanisms [18] have proposed three **orthogonal range search** approaches, including naïve, space and data-based approach. Naïve approach is to approximate **orthogonal search** by external search circle of iDistance. The space approach is mainly to break the **orthogonal range search** into multiple squares. A data-based approach has been proposed to support **orthogonal range search** by iDistance index, better than naïve and space.

iDistance is a relatively classic high-dimensional data index structure, which is based on B+-Tree. This structure, suitable for database index, can reduce the overhead of IO, and filter out a large amount of irrelevant data, thus speeding up retrieval. After IDistance was proposed, it has been applied in large-scale image retrieval [19-21], video retrieval [22], mobile computing [23], distributed systems [24] and other fields.

iDistance algorithm divides the vector space into subspace of clustering by pivot, BC-iDistance [25] algorithm divides the subspace into 2 partitions in each dimension. FGBC-iDistance [26] proposed a more fine-grained partition algorithm and index structure, each part corresponded with a unique FGBC code (fine-grained bit code), which realizes the candidate sets filtered more precisely. M-Chord [24] (iDistance and chord), R-Chord [27] (BC-iDistance and chord), MCAN [28] (iDistance and CAN). Although researchers have proposed a large number of optimization algorithms based on the iDistance algorithm, they just increase the filtering effect of the algorithm, or reduce the calculation time. So far, all researchers have not considered supporting the **orthogonal range search** in the FGBC-iDistance algorithm.

We proposed the finer data decomposition based on FGBC code, results show our novel FGBC (or BC) method generally performs superior even when many wide dimensions are specified, satisfying any practical wildcard searching.

Inspired by the successes of data-based approach in orthogonal range search, our main contributions are as follows:

(1) **A fine-grained data decomposition** called FGBC-iDistance is proposed. The partition strategy divides the cluster subspace of iDistance more meticulously and encodes each space with FGBC code.

(2) **Orthogonal range search** method of FGBC-iDistance is proposed to reduce the intersection of the range of orthogonal search and the space. Results show our novel FGBC (or BC) method generally performs superior even when many wide dimensions are specified.

This paper is organized as follows. Section 2 will briefly overview background and related work to three orthogonal range search approaches, including naïve, space and data-based approach. Section 3 will then cover the orthogonal range search of FGBC-iDistance. Then we present empirical evaluations and discussion in Section 4, and the paper concludes in Sectuon 5.

2 Orthogonal Range Search Approach

This section reviews the three methods based on orthogonal range search. The relevant notations are shown in Table1.

Table 1. Notations

Symbol	Description
$\min(Q_i)$	The minimum value for the range in the *i*th dimension, *v*min
$\max(Q_i)$	The maximum value for the range in the *i*th dimension, *v*max
D	The number of dimensions
Q_i	The tuple representing the min and max values which specify the range in dimension *i*, R_i = <*v*min, *v*mfax>
r	The radius of the range query
c	The center of the search sphere for the naïve solution
X_i^b	The number of divisions for dimension *i* with log base *b*
σ_{ib}	The ratio of a range in dimension *i* (R_i) divided by X_i^b to λ
λ	The value of a side of the hypercubes
d_i	The *i*-th dimension
q	The search vector of *NN* range
P_i	The pivot of *NN* range query (or orthogonal range search)
Ci	The cluster of NN range query (or orthogonal range search)

2.1 Naïve Approach

Literature proposes an orthogonal range search method in the iDistance index, its main idea is to approximate orthogonal search by external query circle of iDistance. Because its performance is unstable, if the "shape" of the orthogonal rectangle is flat in 2 dimensional space, when the difference between the width (the first dimension) and the high (the second dimension) is large, the circle and the rectangle have large area error, which leads to the increase of the candidate set. This is especially problematic for wildcard-capable queries or several dimensions with wide ranges which can expand the query sphere to nearly the size of the entire data-space. Naïve Method algorithm is showed as Algorithm 1:

Algorithm 1. The Naïve Method

```
Input: $Q$ is the range of the orthogonal search.
Output: A single search point and search radius.
1. For each dimension $d_i \in D$ do:
   Calculate the value of $c_i$
       $c_i = \frac{\min(Q_i)+\max(Q_i)}{2}$
2. Calculate radius $r$ using $c$
       $r^2 = \sum_{i=1}^{D} (\max(Q_i) - c_i)^2$
3. Return $c$, $r$ to run in iDistance.
```

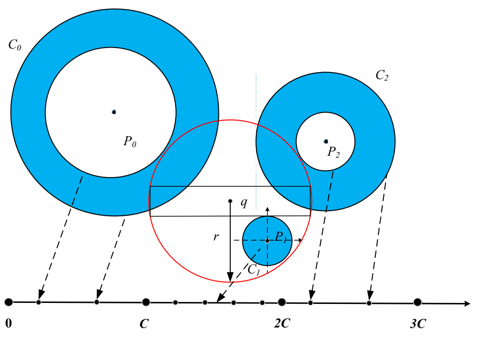


Fig. 1. Orthogonal range search of naïve method

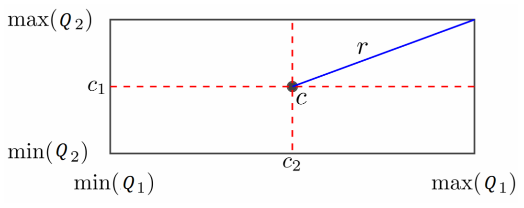


Fig. 2. Naïve query method

2.2 Space-based Decomposition

The space method is mainly to break the orthogonal range search into multiple squares. Each square corresponds to a subquery approximate with external query circle of iDistance. The flat orthogonal rectangle could decompose into squares with unbiased shape, to reduce the candidate set in naïve method. However, Contiguous square circles will cover each other partially, leading overlapping coverage.

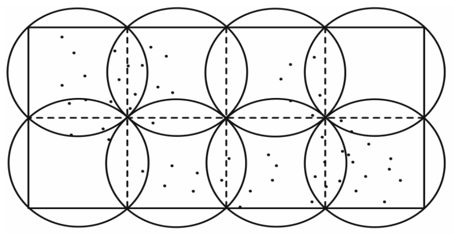


Fig. 3. Space-based decomposition

The space-based method has two ways to decompose: fixed range search and modifying the range search. Taking fixed range query as an example: Based on a $\lambda > 0$, find the optimal number of divisions X_i, for a given dimensional range Q_i, we can use a different λ for each dimension to achieve the best ratio nearest to one. This decomposition will result in less area searched overall, although the complexity will be higher due to the algorithm overhead and combination of multiple partially overlapping (and the number exponentially growing in dimensionality) kNN search.

Algorithm 2. The Space-based Method (Fixed Range)

```
Input: $Q$ is the set of ranges in each dimension.
Output: A set of points and a radius.
1. Select $\lambda = \min(Q)$
2. For each dimension $d_i \in D$ do:
   (a) Calculate all $x_i^b$ for $b = 2, 3, 5, 6$ and $1 \le i \le D$
       $$X_i^b = \log_b(\frac{\| Q_i \|}{\lambda}), b = 2, 3, 5, 6, ...$$
   (b) Choose $b$ based on which base gives the
       $$\sigma_{ib} = \frac{\| Q_i \|}{\lambda(b^{X_i^b})}$$ closest to 1
3. Set the radius based on the chosen $b$ and $X_i^b$ for each dimension
4. Return the centers of the hyper rectangles and the radius.
```

2.3 Data-based Approach

(1) iDistance - indexing technique for vector domains

A data-based [18] approach has been proposed to support orthogonal range search by iDistance index. The iDistance is an essential *k*NN index structure that effectively supports *k*NN and range query, by reducing the high dimensional vector to 1 dimension.

IDistance divides the vector space into cluster subspaces by pivot, avoiding a lot of distance calculations. Firstly, several pivots are selected, and each pivot P_i corresponds to a cluster subspace, as shown in Fig. 3. Each vector is divided into the pivot cluster subspace closest to the vector. The distance between the vector and the anchor maps to the one-dimensional key value iDistance. All vector key values iDistance are managed by B+-tree. Among them, the calculation formula of the key value iDistance is:

$$iDist(x) = dist(P_i, x). \tag{1}$$

Where x is an arbitrary vector and P_i is a pivot, dist() is a Euclidean distance function, and iDist() is a one-dimensional key value function. As shown in Fig. 3, P_0、P_1 and P_2 are pivotes; C_i is the distance of the vector farthest from the pivot P_i in the vector subspace of the pivot P_i, that is, the radius of the vector subspace of the pivot P_i; c is a Constant, greater than all C_i.

Given a range query $Range(q, r)=\{x \in D, dist(q, x) < r\}$, where D is a vector set, the function $dist(q, x)$ represents the distance of the query vector q to any vector x. The process of iDistance range query is: determining whether the search circle of q intersects the vector subspace of the pivot P_i, and if there is no intersection, there is no target vector; if intersecting, the range of the searched ring body needs to be determined, as the blue ring body showed in Fig. 4. Thus, the iDistance range on the axis of the one-dimensional key value is determined, such as [x1-x2], etc.; the distance between each vector and q in the ring body is calculated, and if the distance is less than r, the final search result set is entered.

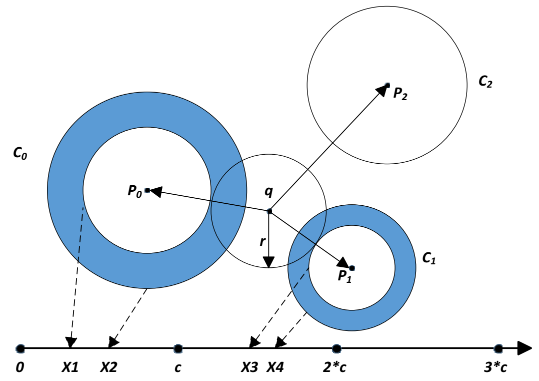


Fig. 4. Range query of iDistance

(2) Orthogonal range search of iDistance

As shown in Fig. 5, P_i is the anchor (center) of cluster C_i, n_i is the point of Q which is closest to P_i, where i is [0, N-1]. N is the number of clusters. The orthogonal range is Q, P_j represents a range interval of Q in the j-th dimension, min(P_j) is the minimum value of the interval and the max(P_j) is the maximum value of the interval, where j means j-th dimensions. There are shadow intersecting rings between Q and C_i which are area of candidates. The shadow intersecting rings of the i-th cluster depend on the value of n_i. n_i could be represented by notation. $\underline{n_i}(j)$ could be calculated by the formula (2).

$$n_i(j) = \begin{cases} \min(Q_j), P_i(j) < \min(Q_j) \\ \max(Q_j), P_i(j) > \max(Q_j) \\ P_i(j),\ \min(Q_j) \le P_i(j) \le \max(Q_j) \end{cases} . \tag{2}$$

Next, the distance $m_i = dist(P_i, n_i)$ between P_i and n_i. The range of i-th shadow intersecting rings is [m_i, r_i], where r_i is radius of the i-th cluster. If m_i is larger than r_i, Q does not intersect the i-th cluster.

The formula 3 for whether the candidates $S(a_1, a_2, \ldots a_d)$ on the iDistince interval [m_i, r_i] satisfy the orthogonal range:

$$S(a_1, a_2 \ldots a_d) \in Q = \{\min(Q_1) \le a_1 \le \min(Q_1)\} \cap \ldots \cap \{\min(Q_d) \le a_d \le \min(Q_d)\}\,. \tag{3}$$

Algorithm 3. Orthogonal range search of data-iDistance

Input: Q is the range of the orthogonal search.
Output: result S
Variables: C_i the set of iDistance cluster, P_i is the center of C_i, r_i is the radius of C_i, O is candidate set.

```
1.  procedure window($Q$)
2.        $S = \phi$, $O = \phi$
3.        for each $1 \le i \le |C_i|$ do
4.            $n_i$ = closetPoint ($Q$, $P_i$)
5.            $m_i$ = dist($P_i$, $n_i$)
6.            if $m_i \le r_i$ then
7.                $O = O \cup$ SearchInB+tree( iDist($m_i$), iDist($r_i$) )
8.            End if
9.        for each $p \in O$ do
10.           if $p \in Q$ then
11.               $S = S \cup p$
12.           End if
13. End.
```

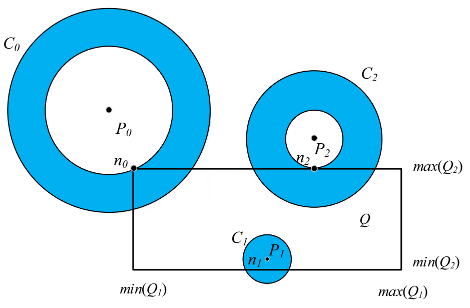


Fig. 5. Orthogonal range search based on data-iDistance

3 Orthogonal Range Search of FGBC-iDistance

3.1 Bit Code

BC-iDistance saves information of both distance and position. As shown in Fig. 6, BC-iDistance adds a BC code to the one-dimensional keyword structure of iDistance, and the BC code divides the pivot into four regions, the BC codes of each region are: 00, 01, 10, and 11, respectively. The vector of the BC code region has a corresponding BC code. When doing the retrieval, the candidate vectors are first selected by the distance value, and then each candidate vector is filtered by the BC code, which is not filtered if the BC code area of the pivot P_i intersects the search circle of q, not filtered if it does not intersect. Therefore, the distance calculation is avoided in the filtered area and the filtering area is more than the filtering area in iDistance. BC-iDistance adds the orientation code on the coordinate axis based on iDistance. The code is composed of binary bit, which is called bit code. By adding a BC code to the one-dimensional keyword structure, and with the characteristics of fast filtering during the filtering of the candidate set, and thus more distance calculation can be avoided. The BC codes of each area are: 00, 01, 10, 11, respectively, that is, BC_0=00, BC_1=01, BC_2=10, BC_3=11, and the vector located in the BC code area has a corresponding BC code.

The granularity of the BC-iDistance bit code is relatively large. For each dimension, it is better to filter only half of the vector for the bit code often intersects with the central axis, because the radius of it is slightly larger, so this dimension loses the filtering effect. According to the filtering effect of the algorithm under various granularities measured by experiments, the paper proposes a bit code filtering method for more fine-grained partitioning of the clustering space. Dividing the space more finely, although it adds some extra space complexity, the filtering effect is better. A BC code filtering decision is added to each vector in the candidate set. The judgment is based on whether the search circle of q and the area of the BC code of a pivot P_i intersect, and if they intersect, the BC code is not filtered, and if they do not intersect, the BC code is filtered. The distance between the data points that are not filtered out and q is calculated. If the distance is less than r, the final search result set is entered.

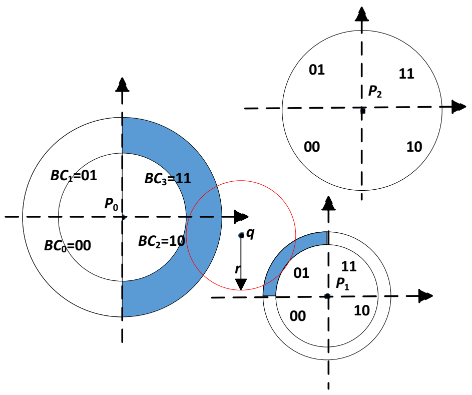


Fig. 6. Range query of BC-iDistance

3.2 FGBC code and Index Structure

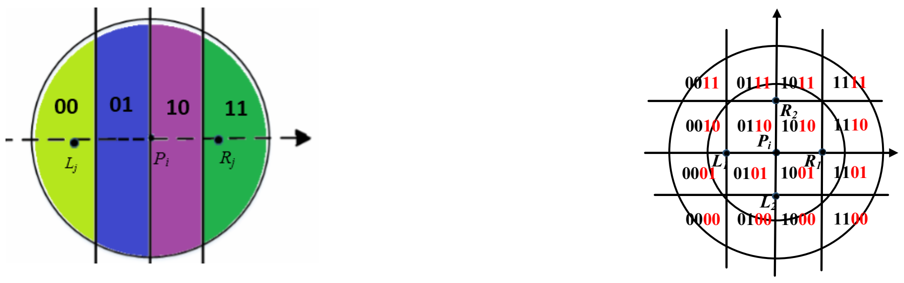
Definition 1. Secondary pivots. The secondary pivots refers to the search for one pivot on each side of each dimension of the pivot P_i (P_1, P_2, ... P_d), and the secondary pivots is represented as ((L_1, R_1), (L_2, R_2), …, (L_d, R_d)), $R_j > L_j$, ($1 \leq j \leq d$), d is the dimension of vector, and j represents the j-th dimension.

Definition 2. Fine-grained bit code (FGBC code). It is assumed that the pivots of the clu2ster subspace to which the vector G belongs are P_i, L_j *and* R_j are two secondary pivots on the j-th dimension of the cluster subspace. Then the FGBC code of G is expressed as B_G ($b_{Gm1}b_{Gm2}$... $b_{Gj1}b_{Gj2}$... $b_{Gd1}b_{Gd2}$). The j-th dimension FGBC code (b_{j1} b_{j2}) of G satisfies the following formula:

$$b_{Gj1}b_{Gj2} = \begin{cases} 11\ , G_j \geq R_j \\ 10\ , P_{ij} \leq G_j < R_j \\ 01\ , L_j \leq G_j < P_{ij} \\ 00\ , \text{otherwise} \end{cases} . \tag{4}$$

As shown in Fig. 7(a), P_i is the pivot, which is the center of the entire cluster subspace, and Lj and Rj are two secondary pivots of P_i in the j-th dimension. Through these three points, each dimension can be divided into four regions, and the FGBC codes of these four regions are 00, 01, 10, and 11, respectively.

The FGBC code divides each dimension of the cluster subspace of the pivot P_i into four regions, so the FGBC code length of each dimension is 2. The length of the FGBC code in the d-dimensional vector space is 2d, and the entire cluster subspace is divided into 2^{2d} regions. As shown in Fig. 7(b), the length of the FGBC code in the two-dimensional vector space is 4 bits, wherein the first two bits of the FGBC code are codes in the first dimension, and the last two bits of the FGBC code are codes in the second dimension. The 4-bit code divides this clustering subspace into 16 small regions.



(a) (b)

Fig. 7. FGBC code of two-dimensional vector space

In Fig. 8, the index structure of FGBC-iDistance is divided into a B+-Tree layer and an FGBC code layer. IO and calculation of Euclidean distance are two time-consuming steps. The space occupied by the FGBC layer is small, and filtering through the FGBC layer can achieve two purposes: One is to reduce the amount of IO access; The second is to reduce the number of distance calculations.

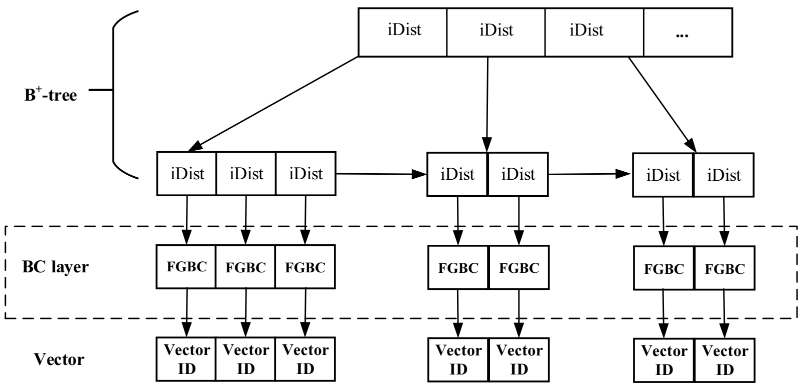


Fig. 8. FGBC-iDistance Index structure

3.3 Principle of FGBC Filtering

Definition 3. The lower bound of the distance of the FGBC code. Suppose q is the query vector, P_i (P_1, P_2, …, P_d) is the pivot, and define the lower bound of the distance of the region S from q to a pivot P_i is minDist (P_i, S, q).

The FGBC codes of the regions q and S are respectively B_q (b_{qm1} b_{qm2} ... b_{qj1} b_{qj2} ... b_{qd1} b_{qd2}), B_S (b_{Sm1} b_{Sm2} ... b_{Sj1} b_{Sj2} ... b_{Sd1} b_{Sd2}). The FGBC codes of q and S are different, and the distance calculation uses the Euclidean distance, then the minDist (P_i, S, q) satisfies the equations (5) and (6):

$$\text{minDist}\left(P_i,S,q\right)\geq\sqrt{\sum_{j=1}^{d}\delta_j^{\,2}}\geq\delta_j. \tag{5}$$

$$\delta_j=\begin{cases}0 & (b_{qj1}b_{qj2}=b_{sj1}b_{sj2})\\ |q_j-R_j| & (b_{qj1}b_{qj2}\neq b_{sj1}b_{sj2}\ \mathit{and}\ b_{qj1}=b_{sj1}=1)\\ & \text{or }(b_{sj1}b_{sj2}=11\ \mathit{and}\ b_{qj1}b_{qj2}=00)\\ |q_j-L_j| & (b_{qj1}b_{qj2}\neq b_{sj1}b_{sj2}\ \mathit{and}\ b_{qj1}=b_{sj1}=0)\\ & \text{or }(b_{sj1}b_{sj2}=00\ \mathit{and}\ b_{qj1}b_{qj2}=11)\\ |q_j-p_{ij}| & (b_{qj1}b_{qj2}=01\ \mathit{and}\ b_{sj1}b_{sj2}=10)\\ & \text{or }(b_{qj1}b_{qj2}=10\ \mathit{and}\ b_{sj1}b_{sj2}=01)\end{cases}. \tag{6}$$

Where δ_j is the distance from the query vector q in the j dimension to region S, q_j is the coordinate of the query vector q in the j-dimension, P_{ij} is the coordinate of the pivot P_i in the j dimension, and L_j is a secondary pivot of the pivot P_i in the j-dimensional coordinates, R_j is the coordinate of the other secondary pivot of the pivot P_i in the j-dimension, $b_{sj1}b_{sj2}$ is the value of the FGBC code of the region S in the j dimension, and $b_{qj1}b_{qj2}$ is the value of the FGBC code of the region where the query vector q is located in the j-dimension.

The correctness of equation (5) is demonstrated below. Suppose $s(s_1, s_2, ..., s_j, ..., s_d)$ is an arbitrary vector in the S region, which is obtained by the Euclidean distance formula:

$$dist(q,s)=\sqrt{\sum_{1}^{d}(q_j-s_j)^2}\,. \tag{7}$$

There are 4 cases existing in $b_{qj1}b_{qj2}$ and $b_{sj1}b_{sj2}$ as follows:

(1) when $b_{qj1}b_{qj2=}$ $b_{sj1}b_{sj2}$, which means that the *j*-dimensional *q* is the same as the FGBC code of *s*, and belongs to the same area in the *j*-th dimension, $(q_j\text{-}s_j)^2 \geq 0$.

(2) when $b_{qj1}b_{qj2} \neq b_{sj1}b_{sj2}$ and $b_{qj1}= b_{sj1}=1$, the value of $(b_{qj1}b_{qj2}, b_{sj1}b_{sj2})$ must be (10,11) or (11,10), q_j and S_j must be on the both sides of R_j, so $(q_j\text{-}s_j)^2 > (q_j\text{-}R_j)^2$; if $b_{sj1}b_{sj2}$=11 and $b_{qj1}b_{qj2}$=00, $q_j < L_j$, $s_j \geq R_j$, however $L_j < R_j$, therefore $s_j\text{-}q_j > R_j\text{-}q_j$, and$(q_j\text{-}s_j)^2 > (q_j\text{-}R_j)^2$.

(3) when $b_{qj1}b_{qj2} \neq b_{sj1}b_{sj2}$&& $b_{qj1}= b_{sj1}=1$, the value of $(b_{qj1}b_{qj2}, b_{sj1}b_{sj2})$ must be (00,01 or (01,00), q_j and S_j must be on the both sides of L_j, so $(q_j\text{-}s_j)^2 > (q_j\text{-}L_j)^2$; if $b_{sj1}b_{sj2}$=00 and $b_{qj1}b_{qj2}$=11, $q_j \geq R_j$, $s_j < L_j$, however $L_j < R_j$, therefore $q_j\text{-}s_j > q_j\text{-}L_j$, and $(q_j\text{-}s_j)^2 > (q_j\text{-}L_j)^2$.

(4) other cases are when ($b_{qj1}b_{qj2}$=01 and $b_{qj1}b_{qj2}$=10) or ($b_{qj1}b_{qj2}$=10 and $b_{sj1}b_{sj2}$=01), q_j and S_j must be on the both sides of P_j, so $(q_j\text{-}s_j)^2 > (q_j\text{-}P_j)^2$.

Since the FGBC codes of *q* and *s* are not the same, at least one dimension meets one of the above (4), (5), (6), based on the above analysis, it can be concluded as:

$$dist(q,s)=\sqrt{\sum_{1}^{d}(q_j-s_j)^2} \geq \sqrt{\sum_{j=1}^{d}\delta_j^{\,2}}\ . \tag{8}$$

Therefore $\operatorname{minDist}\left(P_i, S, q\right) \geq \sqrt{\sum_{j=1}^{d}\delta_j^{\,2}} \geq \delta_j$, and formula (5) is proved.

The formula for judging the search circle of *q* and the region *S* of a pivot P_i is: minDist $(P_i, S, q) < r$, as long as the lower bound of the distance is greater than *r*, the vector of the region *S* can be completely filtered out.

3.4 Filtering Algorithm of FGBC

In the *d*-dimensional vector space, there are 22d kinds of FGBC codes, the time complexity of distance calculation increases exponentially with the dimension d. obviously, filtering cannot be done by calculating the lower bound of distance. In practical applications, it is filtered in paper with the distance difference of the same dimension.

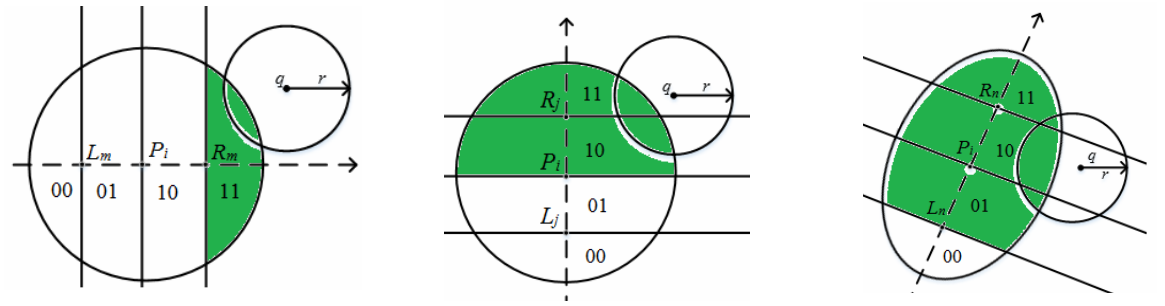
As shown in equation (5), δ_j is the distance between the query vector *q* and the region *S* in the *j*-dimension, and in the *j*-dimension, $minDist(P_i, S, q) = \delta_j$. When $\delta_j > r$, that is $minDist(P_i, S, q) > r$, all vectors on this FGBC code region can be filtered out.

The filtering process of *NN Range*(*q*, *r*): Assume that the query vector $q(q_1, q_2, ..., q_d)$ intersects with a cluster subspace, and the pivots of the cluster subspace are $P(p_1, p_2, ..., p_d)$, the secondary pivots of the cluster subspace are $((L_1, R_1), (L_2, R_2), ..., (L_d, R_d))$, $s(s_1, s_2, ..., s_d)$ is any vector of the cluster subspace. The FGBC codes of (*q*, *s*) are respectively $B_q(b_{q11}\ b_{q12}\ ...\ b_{qj1}\ b_{qj2} ...\ b_{qd1}\ b_{qd2})$, $B_s(b_{s11}\ b_{s12}\ ...\ b_{sj1}\ b_{sj2}\ ...\ b_{sd1}\ b_{sd2})$. For the *j*-dimension, the conditions that the vector *s* can be filtered are as shown in Table 2.

Table 2. Filtering condition of FGBC code of the *j*-dimension vector *s*

$b_{qj1}b_{qj2}$	The query vector *q* satisfies the condition	Filtering condition of vector *s*
11	$q_j\text{-}R_j \geq r$	$b_{sj1}\ b_{sj2} \neq 11$
11	$q_j\text{-}R_j < r$ and $q_j\text{-}p_j \geq r$	$b_{sj1} = 0$
11	$q_j\text{-}R_j < r$ and $q_j\text{-}L_j \geq r$	$b_{sj1}\ b_{sj2} = 00$
10	$q_j\text{-}p_j \geq r$ and $R_j\text{-}q_j > r$	$b_{sj1}\ b_{sj2} \neq 10$
10	$q_j\text{-}p_j \geq r$ and $R_j\text{-}q_j \leq r$	$b_{sj1} = 0$
10	$q_j\text{-}p_j < r$ and $q_j\text{-}L_j \geq r$ and $R_j\text{-}q_j > r$	b_{sj1}^b_{sj2}=0 (That is $b_{sj1}b_{sj2}$ = 00or11)
10	$q_j\text{-}p_j < r$ and $q_j\text{-}L_j \geq r$ and $R_j\text{-}q_j \leq r$	$b_{sj1}\ b_{sj2} = 00$
10	$q_j\text{-}L_j < r$ and $R_j\text{-}q_j > r$	$b_{sj1}\ b_{sj2} = 11$
01	$q_j\text{-}L_j \geq r$ and $p_j\text{-}q_j > r$	$b_{sj1}\ b_{sj2} \neq 01$
01	$q_j\text{-}L_j < r$ and $p_j\text{-}q_j > r$	b_{sj1}=1
01	$q_j\text{-}L_j \geq r$ and $p_j\text{-}q_j \leq r$ and $R_j\text{-}q_j > r$	b_{sj1}^b_{sj2}=0 (That is $b_{sj1}b_{sj2}$ = 00 or11)
01	$q_j\text{-}L_j < r$ and $p_j\text{-}q_j \leq r$ and $R_j\text{-}q_j > r$	$b_{sj1}\ b_{sj2} = 11$
01	$q_j\text{-}L_j > r$ and $R_j\text{-}q_j \leq r$	$b_{sj1}\ b_{sj2} = 00$
00	$L_j\ \text{-}q_j \geq r$	$b_{sj1}\ b_{sj2} \neq 00$
00	$L_j\ \text{-}q_j < r$ and $p_j\text{-}q_j \geq r$	$b_{sj1} = 1$
00	$p_j\text{-}q_j < r$ and $R_j\ \text{-}q_j > r$	$b_{sj1}\ b_{sj2} = 11$

The filtering effect for any dimension can be divided into three categories: the first category, which filters out 3 regions, is as shown in Fig. 9(a), which has the best filtering effect and should be filtered first; the second category filters out 2 blocks of region, as shown in Fig. 9(b), of which the filtering effect is better, so it is secondary; the third category, filtering out one area, is as shown in Fig. 9(c), the priority of this dimension is third. It is better to first determine the dimension with beneficial filtering effect, so as to reduce the running time of filtering.



(a) the *m*-dimension (b) the *j*-dimension (c) the *n*-dimension

Fig. 9. Filtered renderings of different dimensions

3.5 NN range Query of FGBC-iDistance

Taking two-dimensional space as an example, FGBC-iDistance divides the cluster subspace into 16 regions, each region corresponding to a unique FGBC code. After filtering in the P_0 clustering subspace, only three regions with FGBC codes of 1110, 1101, and 1100 are left. After filtering in the P_1 clustering subspace, only two regions with FGBC codes are of 0011 and 0010 are left. The P_2 clustering subspace is completely filtered out.

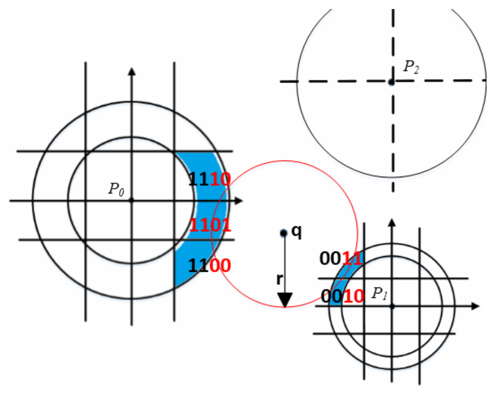


Fig. 10. FGBC-iDistance filtering effect of two-dimensional space

NN range query of FGBC-iDistance is as follows:

Algorithm 4. NN range query of FGBC-iDistance

```
Input: $q$ is the range of the NN query.
Output: result $S$
Variables: $C_i$ the set of iDistance cluster, $P_i$ is the center of $C_i$, $r_i$ is the radius of $C_i$, $O$ is candidate set.
           FG$BC_i$ is the code that $q$ intersects with $C_i$
(1) Determine whether the search circle of $q$ intersects the vector subspace of the pivot $P_i$.
    The judgment formula for intersection is:
        $dist(q, P_i) < C_i + r$.
    The judgment formula for non-intersection is:
        $dist(q, P_i) > C_i + r$.
```

(2) If there is no intersection, there is no target vector in the vector subspace of the pivot; if intersecting, the range of the searched ring body is determined. The formula for calculating the ring body range is:

$$\{x \in P_i \mid \max(\mathrm{dist}(P_i, q)\text{-}r, 0) < dist(p_i, x) < \min(dist(P_i, q) + r, C_i)\}.$$

(3) Determining the search range of iDist, so that a fast search is performed on the B+ tree, and the found vector enters the candidate set. The search range of iDist is:

$$\{x \in P_i \mid i{*}c + \max(\mathrm{dist}(P_i, q)\text{-}r, 0) < iDist(p_i, x) < i{*}c + \min(dist(P_i, q)+r, C_i)\}.$$

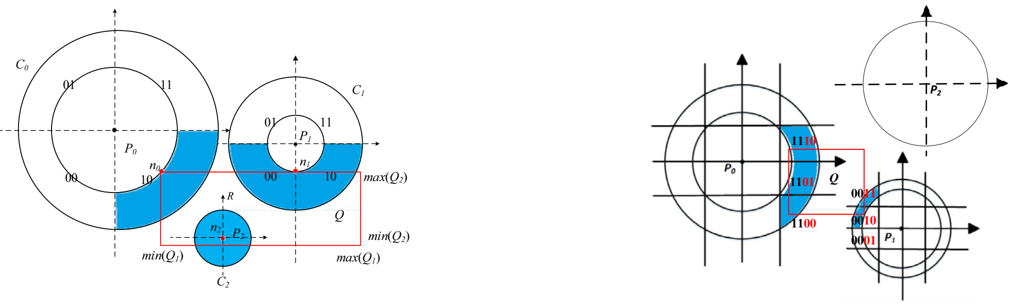
(4) Performing FGBC code filtering on each vector in the candidate set. The principle of judging whether to filter is: whether the search circle of *q* and the area of the FGBC code of a pivot P_i intersect, and if they intersect, they are not filtered, and if they do not intersect, they are filtered.

(5) Calculating the distance between *q and* each vector in the filtered candidate set. If the distance is less than r, enter the final search result set.

Distance calculation is a time-consuming operation. In d-dimensional space, the calculation number of distance of BC-iDistance range query is better to be reduced to $1/2^d$ of iDistance, and the worst is the distance calculation of iDistance. The calculation number of distance of FGBC-iDistance is better to be reduced to $1/2^{2d}$ of iDistance, and the worst is the number of distance calculations of BC-iDistance. The comparison in the number of distance calculations is: FGBC-iDistance ≤ BC-iDistance ≤ iDistance.

3.6 Orthogonal Range Search of FGBC-iDistance

This filtering with BC code (or FGBC code) has fewer candidate sets. How to judge *Q* intersects with clusters? As shown in Fig. 11(a), take cluster C_0 as an example. In the first dimension of $\min(Q_1) > P_0(1)$, the BC of intersection between *Q* and C_0 in the first dimension is 1. In the second dimension of $\max(Q_2) < P_0(2)$, the BC of intersection between *Q* and C_0 in the second dimension is 0. The intersecting rings of cluster C_0 reduce to 1/4 of the ring with *BC* =10. And so on, the intersecting rings of cluster C_2 reduces to semi-ring with *BC* =00 or 10.



(a) Orthogonal Range Search of BC-iDistance (b) Orthogonal Range Search of FGBC-iDistance

Fig. 11.

Algorithm 5. Orthogonal Range Search of FGBC-iDistance

Input: *Q* is the range of the orthogonal search window.
Output: result *S*
Variables: C_i the set of iDistance cluster, P_i is the center of C_i, r_i is the radius of C_i, *O* is candidate set. BC_i is the code that *R* intersects with C_i

```
procedure window($Q$)
        $S = \phi$, $O = \phi$
        for each $1 \le i \le |C_i|$ do
            $n_i$ = closetPoint ($Q$, $P_i$)
            $m_i$ = dist($P_i$, $n_i$), $BC_i$ or $FGBC_i$ = GetBCArea($Q$, $P_i$)
            if  $m_i \le r_i$  then
                $O = O \cup$ SearchInB+tree( iDist($m_i$), iDist($r_i$), $BC_i$)
            End if
        for each $p \in O$ do
```

```
10.            if $p \in Q$ and fgbc = $FGBC_i$ then
11.                $S = S \cup p$
12.            End if
13. End.
```

4 Experimental Evaluations on Orthogonal Range Search of FGBC-iDistance

Main evaluating indicator. In fact, disk-based I/O bottlenecks are a common problem with inefficient retrieval methods. Therefore, it is more important in experiments that the index properly filter out data points (fewer candidates) so that they do not have to be accessed. More attention should be paid to candidates, and query time between the different indexing methods.

- **Candidates (%,** is called **CD)**: The proportion of candidate set to the total set of data;
- **Query Results (%,** is called **QR):** The proportion of correct query result set to the total set of data;
- **Query Time (ms**, is called **QT):** The consuming time of taking for the final correct query sets;
- **Nodes:** The number of accessing the B+-tree Nodes;
- **Selectivity:** Query selection rate which is measure size of the orthogonal range search area.

Datasets.

(1) Synthetic Dataset

Considering that the real world data set is generally between the uniform distribution and the Gaussian distribution, the selected synthetic dataset which includes uniformly distributed data and Gaussian distributed data, make the experimental results more realistic. Each composite data set includes 100,000 data. The values on each dimension of the uniformly distributed data are randomly sampled from 0 to 1, and the Gaussian distribution data is generated using Matlab and contains 16 Gaussian distribution data sets. We set the default dimension to 10 and set the default variance of each Gaussian distribution between 0.2 and 0.4.

(2) Real Dataset

As shown in Table 3, we select 3 real datasets whose distribution are very different in space, for providing a more comprehensive assessment, and demonstrating the differences on performance of these methods in different types of real data. MUSIC dataset has the densest data distribution. Most datasets are concentrated in a small area of the entire space. The COLOR datasets have slight overlap in the entire cluster space. SIFT data is sparsely distributed with little overlap. The distribution of the COLOR dataset data is between the MUSIC dataset and the SIFT dataset.

Table 3. Real dataset

dataSet	numbers	dimensions	name	further
COLOR	68040	32	UCI core COLOR histograms	well-distributed
MUSIC	515345	90	UCI song year prediction	dense
SIFT	1000000	128	ANN_SIFT1MIL	sparse

(3) Methods

As shown in Table 4, we select five orthogonal range search methods for comparison. The five orthogonal range search methods of wide dimension's query are referred, as naïve-w, space-w, data-w-idist, data-w-BC, data-w-FGBC.

Table 4. Five orthogonal range search methods for comparison

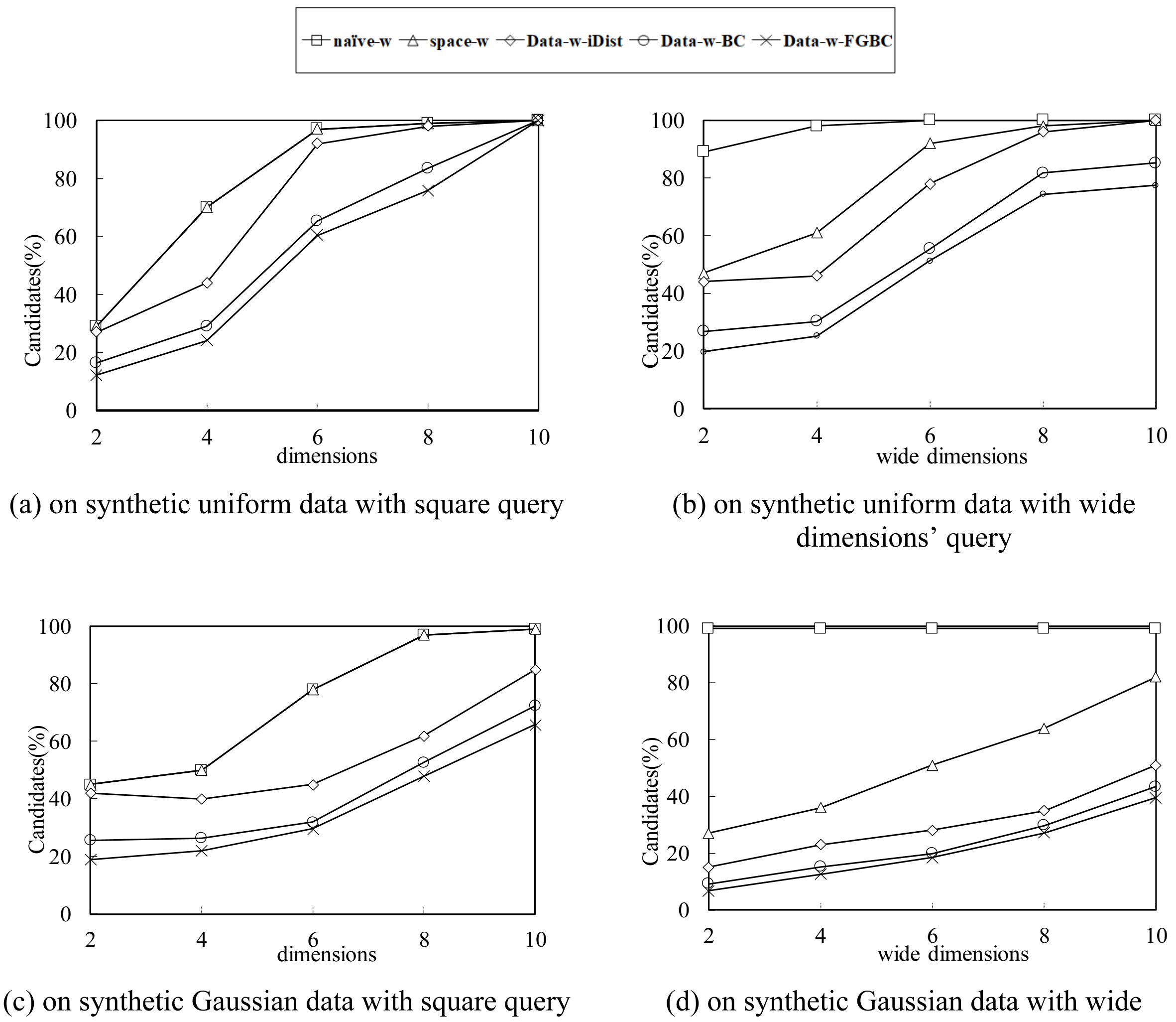
method	name	section
naïve	Algorithm 1	Section 2.1
space	Algorithm 2	Section 2.2
data-idist,	Algorithm 3	Section 2.3
data-BC	Algorithm 5 (BC)	Section 3.6
data-FGBC	Algorithm 5 (FGBC)	Section 3.6

4.1 Synthetic Dataset Comparison

Selectivity describes the measure of the size of the orthogonal range search area, whose value is equal to the size of the query range divided by the size of the entire data space. For example, the range on each dimension in the two-dimensional space is [0, 1], and the orthogonal range search window(Q) = {[0.1, 0.6], [0.2, 0.7]}, Then the query selection rate is 25%. When the query range of an orthogonal range search on a dimension is [0, 1], then this dimension is called a wide dimension.

4.1.1 Analysis of CD with Different Distributions

The dimension is set to 10 dimensions, and the query selectivity of the orthogonal range search is set to 1%. As shown in Fig. 12(a) and Fig. 12(b) on synthetic uniform data and Fig. 12(c) and Fig. 12(d) on synthetic Gaussian data, candidates change with 2 to 10 dimensions. The above five methods were compared in the orthogonal square range query (naïve, space, data-idist, data-BC, data-FGBC) and in the orthogonal wide dimension's query (naïve-w, space-w, data-w-idist, data-w-BC, data-w-FGBC). For a square query, the range is the same (max(Q_j)-min(Q_j)) in each dimension; The number of wide dimensions is set to half of the total dimension.



(a) on synthetic uniform data with square query

(b) on synthetic uniform data with wide dimensions' query

(c) on synthetic Gaussian data with square query

(d) on synthetic Gaussian data with wide dimensions' query

Fig. 12. The candidates varying on dimensionality compared with range query methods

For both types of synthetic data, we have seen that the naïve method is difficult to handle a wide range of queries (naïve-w). Solutions based on spatial decomposition and data decomposition-based methods are better than naïve methods. Based on the data decomposition method, the range of intersecting rings is the most accurate. At the same time, the BC code and the FGBC code are used to streamline the scope of the ring. The overall effect is better than other methods and is considered as the best method.

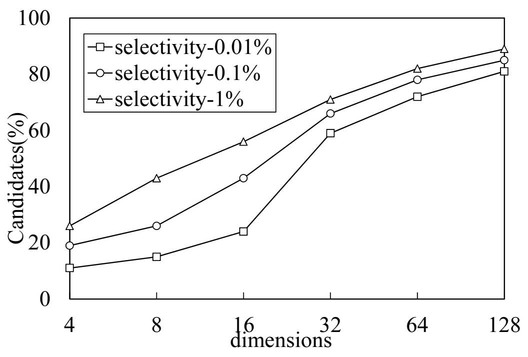
4.1.2 Analysis of Different Selectivity

We need to focus on the impact of different selectivity on experimental results. We adjust the variance to 0.1-0.2, to ensure more cluster separation, the dimension is set to 128, a data-BC index is established. The 3 different selectivities of 1%, 0.1%, and 0.01% were used to conduct orthogonal range search tests based on BC code data based decomposition. The analysis is as follows:

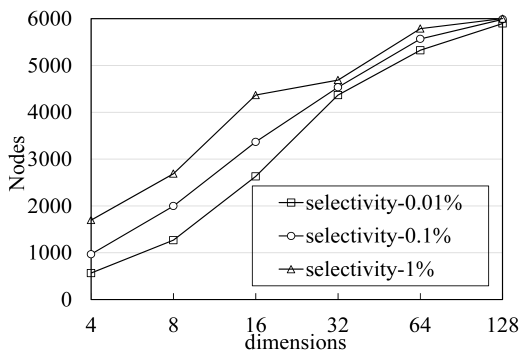
(1) As shown in Fig. 13(a) and Fig. 13(b), the trend of the CD is similar to the trend of the Nodes. In theory, under certain dimensions, the larger the selectivity is, the greater the CD is, and the more the Nodes are. Between 2 and 32 dimensions, the CD is the smallest with 0.01% of smallest selectivity.

(2) Fig. 13(c) shows that the QT increases along with the BC filtering time increases, simultaneously the filtering intersection between the orthogonal range search area and the BC regions increases with the rise of dimension On the other hand, the QT also increases, along with the increase of query selection rate which cause more candidates returned.

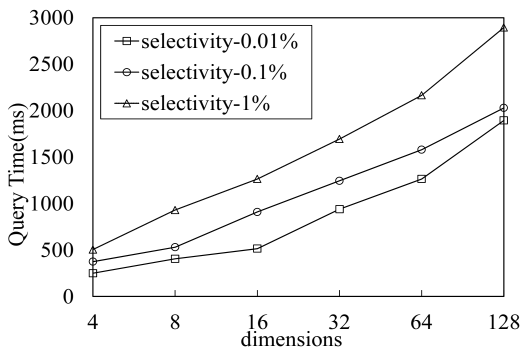
(3) Fig. 13(d) is the final result set to the size of the entire data set. For the same query selectivity, QR increases, because that the range of the query on a single dimension increases with the dimension rises.



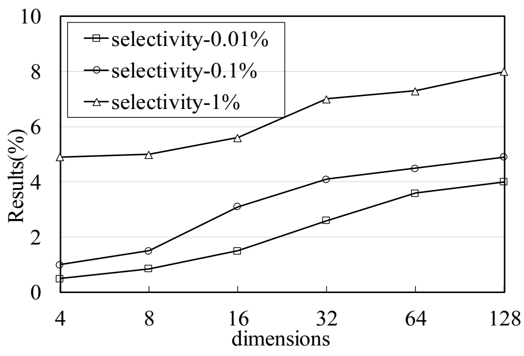
(a) Candidate sets at different selectivity



(b) Number of node visits at different selectivity



(c) Query time under different selectivity



(d) Result set at different selectivity

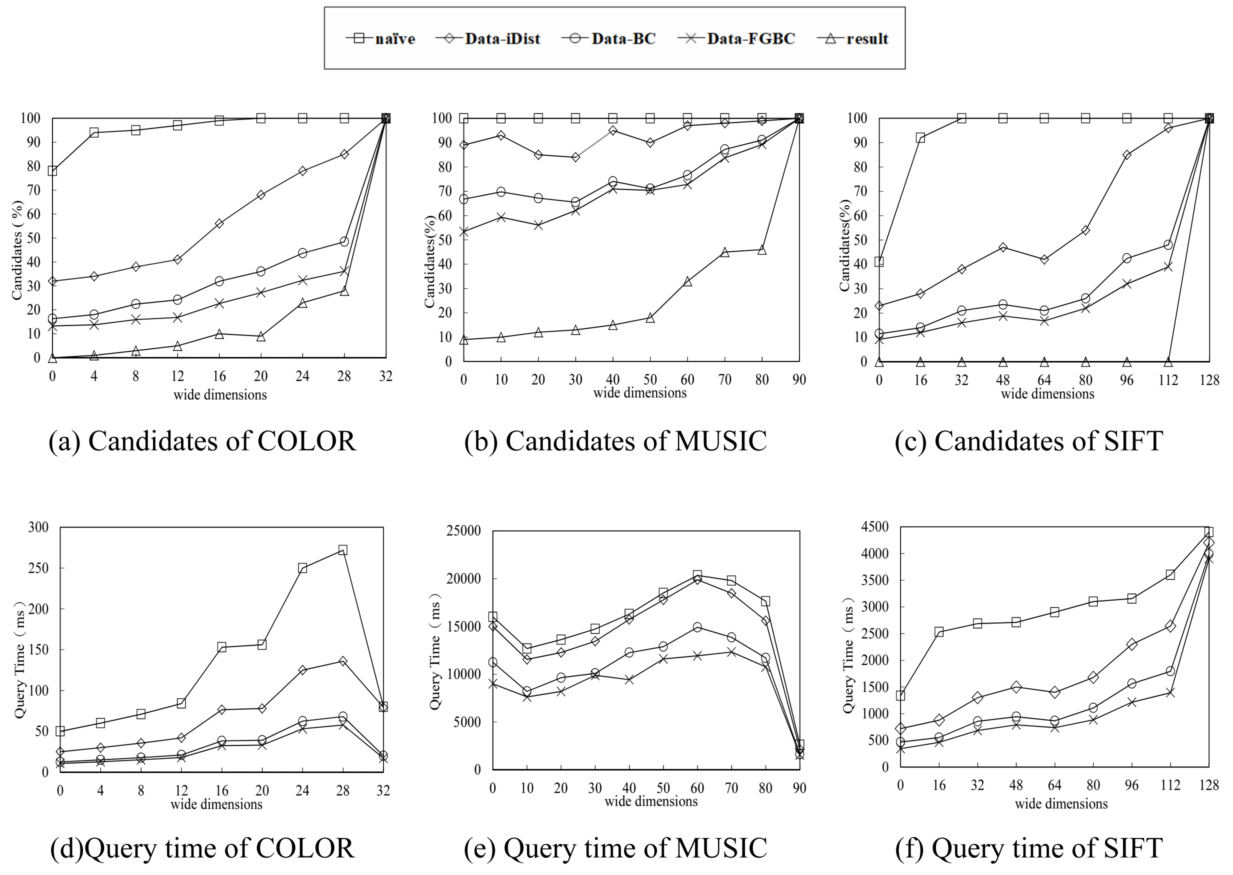
Fig. 13. Orthogonal range search on different dimensions and selectivity

4.2 Real-world Dataset Comparison

We test the results of orthogonal range queries in different width dimensions, including the candidate set and query time. For space, we omit results for B+-tree nodes accessed, as they directly mimic candidates accessed for all tests performed, which is typical behavior. We also omit results for the space-based decomposition method because while reasonably effective in low dimensions, or with only a few wide dimensions, it (as expected) quickly becomes inefficient and impractical in higher dimensions. As shown in Fig. 13, there are following conclusions.

4.2.1 QT is Directly Proportional to CD

The most important factor here is the filter capability which depends mainly on the size of CD. In most cases of all data sets, QT is directly proportional to CD. In a particular case of full dimension, less time is required when the entire space is forced to be searched upfront (all wide dimensions). It is specific in 32 dimensions in Fig. 14(a) and Fig. 14(d) and concrete in 90 dimensions in Fig. 14(b) and Fig. 14(e), QT reaches the minimum value, with CD maximum. However, in SIFT of Fig. 14(c) and Fig. 14(f), QT is directly proportional to CD, that is because the largest number of SIFT datasets is sparse, resulting in very small result sets size for more time which is required for the entire space with the largest number, and reaching the peak value under all wide dimensions, no matter what kind of decomposition method (data-BC or data-iDist).



(a) Candidates of COLOR (b) Candidates of MUSIC (c) Candidates of SIFT

(d)Query time of COLOR (e) Query time of MUSIC (f) Query time of SIFT

Fig. 14. Performance comparison of Real dataset

4.2.2 Performance Comparison of Methods

In COLOR of Fig. 14(a) and Fig. 14(d), CD of Data-iDist probably accounts for between 70% and 30% within 20 wide dimensions of the query, CD is reduced by about half by BC code filtering that CD of Data-BC accounts for between 35% and 15%, but QT of Data-BC is reduced to about 65% of Data-iDist, and not as much as half. The reason is that there is also an increase in the time required for BC code filtering. In sparse SIFT of Fig. 14(c) and Fig. 14(f), the Data-BC performs improved obviously than the Data-iDist. In MUSIC of Fig. 14(b) and Fig. 14(e), the Data-BC only performs slightly better than the Data-iDist. Because in MUSIC of dense data sets, the return result of any query is over 20%. This means that the ratio of data decomposition to reduce CD is much less than that of COLOR.

In COLOR of Fig. 14(d), we see that less time is required when the entire space is forced to be searched upfront (all wide dimensions), rather than methodically determining to search most of the space anyway (e.g., 28 wide dimensions). The most important factor here is the filter capability, and we see the Data-iDist method gracefully degrades to the naïve performance as the number of wide dimensions increases.

Notice in Fig. 14(b) that the initial query already returns about 20 % of the dataset, which could be affecting relative performance comparisons. For each dataset we pick the query points and wide dimensions in the same random manner, so it seems to imply a difference in dataset characteristics.

The results, presented in Fig. 14(c) and Fig. 14(f), are quite different than the Music dataset, rejecting any purely "high-dimensional" performance factors in favour of more rich factors tied closely to the dataset characteristics. Here we see exceptional performance for the Data-FGBC method. However, it is likely in part due to the extremely small result set size, which on average is only the data point the query is centered on. To our surprise, this holds true even with almost all wide dimensions in the query. This explains why the time is taken increases with all wide dimensions (the opposite of other datasets) because it finally incurs a higher cost during the refinement step. By maintaining relatively high performance in retrieving a small set from a large and high-dimensional dataset, this suggests a promising application for retrieving highly selective high-dimensional range queries.

5 Conclusions

This paper presented and analyzed five possible approaches for executing orthogonal range (window) search in original indexes for *k*NN search. We agree on data decomposition to reduce the intersection of the range of orthogonal query and the space. While we proposed the finer data decomposition based on FGBC code, results show our novel FGBC (or BC) method generally performs superior even when many wide dimensions are specified, satisfying any practical wildcard searching.

Acknowledgements

We are grateful to the support of Major Project for New Generation of AI Grant No. 2018AAA0100400, research fund of Hunan Province (2018JJ2099, 2018JJ2098, 19C0558, CX1911), Hunan open fund (Project Similarity detection cloud platform; Clothing material aggregation and prediction), Hunan Education Reform fund (The whole process performance management and evaluation system of innovation and entrepreneurship center).

References

[1] C.-Q. Wu, P.-G. Ren, X.-F. Wang, Survey on semantic-based organization and search technologies for network big data, Chinese Journal of Computers 38(2015) 1-17.

[2] A. Guttman, R-trees: a dynamic index structure for spatial searching, ACM 14(2)(1984) 47-57.

[3] J.-L. Bentley, Multidimensional Binary Search Trees Used for Associative Searching, Communications of the ACM 18(9)(1975) 509-517.

[4] V. Gaede, Multidimensional access methods, ACM Computing Surveys 30(2)(1999) 170-231.

[5] W.-C. Fu, M.-S. Chan, Y.-L. Cheung, Dynamic vp-tree indexing for n-nearest neighbor search given pairwise distances, The VLDB Journal 9(2)(2000) 154-173.

[6] P. Ciaccia, M. Patella, P. Zezula, M-tree: An Efficient Access Method for Similarity Search in Metric spaces, The VLDB (1997) 426-435.

[7] H.-V. Jagadish, B.-C. Ooi, K.-L. Tan, iDistance: an adaptive b+-tree based indexing method for nearest neighbor search, ACM Transactions on Database Systems 30(2)(2005) 364-397.

[8] H.-N. Gabow, J.-L. Bentley, R.-E. Tarjan, Scaling and related techniques for geometry problems, in: Proc. ACM Symposium on Theory of Computing, 1984.

[9] B. Chazelle, A functional approach to data structures and its use in multidimensional searching, SIAM Journal on Computing 17(3)(1988) 427-462.

[10] M.H. Overmars, Efficient data structures for range searching on a grid, Journal of Algorithms 9(2) 254-275.

[11] S. Alstrup, G.-S. Brodal, T. Rauhe, New data structures for orthogonal range searching, in: Proc. Foundations of Computer Science, 2000.

[12] Y. Nekrich, Orthogonal range searching in linear and almost-linear space, Computational Geometry: Theory and Applications 42(4)(2009) 342-351.

[13] M. Lewenstein, Orthogonal range searching for text indexing, in: Proc. Space-Efficient Data Structures, Streams, and Algorithms, 2013.

[14] K. Ishiyama, K. Sadakane, A succinct data structure for multidimensional orthogonal range searching, in: Proc. 2017 Data Compression Conference (DCC), 2017.

[15] T.-M. Chan, Orthogonal range searching in moderate dimensions: KD trees and range trees strike back, Discrete & Computational Geometry 61(4)(2019) 899-922.

[16] T.-M. Chan, K. Tsakalidis, Dynamic orthogonal range searching on the ram, in: Proc. 33rd International Symposium on Computational Geometry (SoCG 2017), 2017.

[17] K. Bringmann, T. Husfeldt, Magnusson. M, Multivariate analysis of orthogonal range searching and graph distances, Algorithmica 82(2020) 2292-2315.

[18] T. Wylie, M.-A. Schuh, R.-A. Angryk, Enabling high-dimensional range queries usingkNN indexing techniques: approaches and empirical results, Journal of Combinatorial Optimization 32(4)(2016) 1107-1132.

[19] Y. Zhuang, F. Wu, Fast answering k-nearest-neighbor queries over large image databases using dual distance transformation, in: Proc. Advances in Multimedia Modeling, 2007.

[20] J. Zhang, X. Zhou, W. Wang, Using high dimensional indexes to support relevance feedback based interactive images retrieval, in: Proc. 32nd International Conference on Very Large Data Bases, VLDB Endowment, 2006.

[21] W. Wang, X.-L. Wang, Wilkes. D. M, Modifying iDistance for a fast CHAMELEON with application to patch based image segmentation, in: Proceedings of the. 9th IASTED International Conference on Signal Processing, Pattern Recognition and Applications (SPPRA 2012), 2012.

[22] H.-T. Shen, B.-C. Ooi, X. Zhou, Towards effective indexing for very large video sequence database, in: Proc. 24th ACM SIGMOD International Conference on Management of Data, Association for Computing Machinery, 2005.

[23] S. Ilarri, E. Mena, Illarramendi. A, Location-dependent queries in mobile contexts: distributed processing using mobile agents, IEEE Transactions on Mobile Computing 5(8)(2006) 1029-1043.

[24] D. Novak, P. Zezula, M-Chord: a scalable distributed similarity search structure, in: Proc. Infoscale Hong Kong, 2006.

[25] J.-J. Liang, Y.-C. Feng, BC-iDistance: an optimized high-dimensional index for KNN processing, Journal of Harbin Institute of Technology 15(6)(2008)856-861.

[26] X.-P. Yuan, C.-F. Wang, J. Long, FGBC-iDistance: fine-grained bit-code filter based high-dimensional index, Journal on Communications 38(Z1)(2017) 127-134.

[27] W. Yin, M. Zhu, L. Jiang, R-Chord: A distributed similarity retrieval system with RPCID, in: Proc. IEEE International Conference on Network Infrastructure and Digital Content, 2009.

[28] F. Falchi, C. Gennaro, P.A. Zezula, Content-addressable network for similarity search in metric spaces, in: Proc. Databases, Information Systems, and Peer-to-Peer Computing, 2007.