Exploring Bit-Difference for Approximate KNN Search in High-dimensional Databases

Bin Cui[1], Heng Tao Shen[2], Jialie Shen[3], Kian-Lee Tan[1]
[1] Singapore-MIT Alliance, National University of Singapore
[2] The University of Queensland Australia, [3] University of New South Wales Australia

***Abstract*— In this paper, we develop a novel index structure to support efficient approximate k-nearest neighbor (KNN) query in high-dimensional databases. In high-dimensional spaces, the computational cost of the distance (e.g., Euclidean distance) between two points contributes a dominant portion of the overall query response time for memory processing. To reduce the distance computation, we first propose a structure (BID) using BIt-Difference to answer approximate KNN query. The BID employs one bit to represent each feature vector of point and the number of bit-difference is used to prune the further points. To facilitate real dataset which is typically skewed, we enhance the BID mechanism with clustering, cluster adapted $bitcoder$ and dimensional weight, named the BID$^+$. Extensive experiments are conducted to show that our proposed method yields significant performance advantages over the existing index structures on both real life and synthetic high-dimensional datasets.**

***Index Terms*— High-dimensional index structure, approximate KNN query, memory processing, bit difference**

I. Introduction

With an increasing number of new database applications such as multimedia content-based retrieval, time series and scientific databases, the design of efficient indexing and query processing techniques over high-dimensional datasets becomes an important research area. These applications employ the so called feature transformation which transforms important features or properties of data objects into high-dimensional points, i.e. each feature vector consists of D values, which correspond to coordinates in a D-dimensional space. Searching for objects based on these features is thus a search of points in this feature space. In these applications, one of the most frequently used and yet expensive operations is to find objects in the database that are similar to a given query object. K-nearest neighbor search is a central requirement in such cases.

There is a long stream of research on solving the k-nearest neighbor search problem, and many multidimensional indexes have been proposed [5], [2], [3], [9], [10], [12], [17], [20], [21]. These techniques focused on getting exact results from queries, where exactness is defined in terms of the feature vectors and a distance function between them. However, the exact answer is highly subjective in some data applications, such as multimedia content-based retrieval application. In this case, the feature vectors themselves are approximate representations of image features such as color, texture and shape. There do not exist a feature extraction mechanism or a distance evaluation method which mimic similarity measurement based on human perception. On the other hand, computing exact nearest neighbors in high-dimensional space is costly because of the expensive distance computation. However, it has been shown that by computing nearest neighbors approximately, it is possible to achieve significantly shorter execution time with a relatively small error ratio, and that users would find approximate answers acceptable with faster response. Approximate KNN query allows the user to select a maximum error factor, thus providing a tradeoff between accuracy and response time. Approximate query-answering techniques have recently received many attentions [14], [19], [8], [15], [18], [1], [11].

These index structures have largely been studied in the context of disk-based systems where it is assumed that the databases are too large to fit in main memory. This assumption is increasingly being challenged as RAM gets cheaper and larger. This has prompted renewed interest in research in main memory databases [6], [13], [16]. As random access memory gets cheaper, it becomes increasingly affordable to build computers with large main memories, and it is possible to store the whole database in memory for faster system response. But main memory data processing is not as simple as increasing the buffer pool size, minimizing L2 cache misses and computation cost has been an active area of research. Especially since the high-dimensional distance computation is CPU intensive, an efficient main memory query mechanism should minimize the distance computation to improve the performance.

In this paper, we propose a novel index structure to support approximate KNN search by exploiting bit mapping and bit-difference, called the BID. Each dimension of the point is represented by a single bit and the bit value is generated by comparing the point coordinates with the mean of the dimensional domain, which is called $bitcoder$. For example, if the domain of each dimension is [0, 1], a bit value is set to 1 if the value is larger than 0.5, and 0 otherwise. Since only one bit is used to code each dimension, the compression of BID file is very efficient, i.e. only $\frac{1}{32}$ of the original file size. Thus the real vectors can be represented by an array of bit-strings. When a query is issued, we generate the bit-

B. Cui, and K.-L. Tan are with Singapore-MIT Alliance, National University of Singapore. Email: {cuibin, tankl}@comp.nus.edu.sg
H.T. Shen is with School of Information Technology & Electrical Engineering, The University of Queensland Email: shenht@itee.uq.edu.au
J. Shen is with School of Computer Science and Engineering, University of New South Wales Australia Email: jls@cse.unsw.edu.au
This paper appeared at the 16th Australasian Database Conference, University of Newcastle, Newcastle, Australia. Conferences in Research and Practice in Information Technology, Vol. 39. H.E. Williams and G. Dobbie, Ed.

string of the query point, and scan the bit-strings of data. The query result is determined by the bit-difference of the query point and data. Using the bit comparison, we can not only reduce the cache misses, but also avoid the expensive distance computation and hence reduce the response time. Note that, although the VA-file [20] also uses bit compression, it has to compute the distance based on the approximation file.

To support real-life datasets which are typically skewed and clustered, we enhance the BID with some modifications and call it the BID$^+$. First, we partition the dataset into clusters, and each cluster has different $bitcoder$ according to the respective data distributions. The coordinates of cluster center are used as $bitcoder$ for each cluster in our implementation. The bit-strings in the same cluster are stored in memory consecutively. Second, the dimensions with larger variance are more important, as the distance in such dimensions generally dominates the overall distance between two points, therefore we evaluate the variance of each dimension and give the dimensions different weights. The BID$^+$ method starts the query from the nearest clusters and thus can avoid scanning the whole bit-strings. To ease the presentation, we consider only the Euclidean distance metric. However the proposed method can be easily extended to support other metrics such as L_1 or L_∞.

We present a detailed experimental evaluation of our proposed BID and BID$^+$, and compare the proposed schemes against some existing index structures. The results show that our methods can handle approximate KNN queries more efficiently for different high-dimensional datasets.

The remainder of this paper is organized as follows. In the next section, we present the approximate quality metrics and review some related work. In Section III, we introduce our newly proposed BID and BID$^+$ structures. We also present the approximate KNN search and construction operations on the structures. Section IV reports the findings of an extensive experimental study conducted to evaluate the proposed schemes. Finally, in section 5 we draw some conclusions and indicate future directions for this work.

II. Related Work

A. Approximation quality metrics

For similarity queries, the quality of the result set is traditionally measured by a combination of two important quality metrics: recall and precision. They can be described as completeness of retrieval and purity of retrieval respectively. Recall is a measure of how well the retrieval method finds all relevant objects, and precision is a measure of how well such a system finds the relevant objects. The irrelevant objects in the result set are called false hits and the relevant objects that are not in the result set are false dismissals. For approximate KNN query, the number of false hits and the number of false dismissals become the same value. In other words, if the approximation causes some false dismissals, these dismissals are replaced by false hits. Computing the number of false hits, or dismissals, is enough to capture the traditional error metric, which we will refer to as the $ratio\ of\ false\ dismissals$ (RFD). We use the metric RFD as one of our metrics in the evaluation of the proposed techniques. Let NN_i, where $i \in [1, K]$, be the i-th nearest neighbor (NN) in the accurate result set, ANN_i be the i-th NN in the approximate result set, Q be the query point, and $Distance\_K$ be the K-th nearest neighbor distance of accurate result. Obviously, an approximate result is good if it contains most NNs of the accurate result, i.e. fewer false dismissal. The $ratio\ of\ false\ dismissals$ can be defined as follows:

$$RFD = \frac{1}{K}\sum_{1}^{K}\begin{cases}1 & distance(ANN_i, Q) > \text{Distance\_K} \\ 0 & otherwise\end{cases} \quad (1)$$

Values of RFD close to 0 indicate a high quality of the result.

However, the value of RFD does not fully capture important information about the quality of the approximations. With this metric, two techniques may get two different answer sets which have the same RFD value, but it cannot differentiate the qualities of these two answer sets where one may include some far away points. Therefore, we also employ a metric which takes into account the quality of the answers with respect to closeness to the query object. This alternative error metric is particularly useful for approximate KNN search, and we refer to it as the $ratio\ of\ distance\ errors$ (RDE) which can be defined as follows:

$$RDE = 1 - \frac{\sum_1^K distance(NN_i, Q)}{\sum_1^K distance(ANN_i, Q)} \quad (2)$$

Values of RDE close to 0 also indicate a high quality of the result.

B. Existing approximate KNN algorithms

Approximate query-answering techniques have recently received many attentions [14], [19], [8], [15], [18], [1]. Below, we will review two state of arts techniques that are used for comparison in our experimental study.

In [19], the authors investigated approximate KNN query evaluation techniques based on the VA-file [20]. The VA-file is a vector approximation scheme for high-dimensional indexing. It divides the data space into a 2^b rectangular cells, where b denotes a user specified number of bits. The scheme allocates a unique bit-string of length b for each cell, and approximates data points that fall into a cell by that bit-string. The VA-file itself is simply an array of these approximations. When searching for the nearest neighbors, the entire approximation file is scanned and the upper and lower distance bounds to the query point Q can easily be determined based on the cell represented by the approximations. After the filtering step, a small set of candidates remain. These candidates are then visited and the actual distances to Q are determined, i.e. the exact KNN answers. To develop approximate query-evaluation techniques, they derive formulae for the distribution of the error of the bounds and the duration of the different phases of query evaluation. Based on these results, two different approximate query evaluation techniques are developed. The first one adapts the bounds to have a more rigid filtering, named VA-BND; the second one skips computation of the exact distances, i.e. the vectors with the K smallest lower bounds are retrieved as the approximate results, named VA-LOW.

More recently, a general framework for approximate nearest neighbor queries was proposed in [8]. The current approaches for nearest neighbor query processing can be categorized based on either their ability to reduce the data set that needs to be examined (retrieved set reduction), or their ability to reduce the representation size of each data object (representative size reduction). The authors first proposed modifications to existing techniques to support the progressive processing of approximate nearest neighbor queries. After that, a new technique was proposed, which effectively combines the two class of approaches into a single framework, i.e. by reducing the size of the retrieved set and feature vector sizes for efficient approximate searching. First, a dimensionality reduction is performed on each data point in the data set. Second, the retrieved portion of data is reduced by the help of a clustering technique, which is an adaptation of K-means clustering method, and the feature vectors within a cluster are organized to support interactive approximate searching.

III. Approaches for Approximate KNN Query

In this section, we give the motivation and develop new index structures and algorithms that facilitate fast approximate KNN search in main memory environment. We start by presenting the basic structure of BIt-Difference (BID). Then we will discuss how it can be improved to support skewed (clustered) dataset.

A. The BIt-Difference (BID)

The VA-file [20] is proposed to deal with queries for high-dimensional databases. The VA-file works well for uniformly distributed data in disk-based environment, because it can reduce disk I/O significantly compared with sequential scan. However the VA-file incurs three cost overheads: the first is decoding cost, because the bit-string must be decoded for distance calculation; the second is computation cost, both upper and lower bounds of the distance to the query point must be determined; the third cost is data access cost as the VA-file needs to access the real data set to get exact distances. Since disk I/O cost is dominant in disk-based environment, these cost overheads do not affect the performance of VA-file much. However, high-dimensional KNN search cost is bounded by CPU cost in main memory system. The higher computational cost incurred by VA-file makes it less attractive for main memory databases. Although the variant VA-LOW proposed in [19] can reduce the computational cost, it has to scan the whole approximation file to compute the low-bound distance.

On the other hand, following [2], we can determine the expected distance of the query point to the actual nearest neighbor in the database. For simplicity, let us consider uniformly distributed data in a normalized data space $[0,1]^D$ having a volume of 1 and the data size is N. The nearest neighbor distance may then be approximated by the volume of the sphere which, on average, contains one data point. The data space with radius r can be calculated by

$$sp^d(r) = \frac{\sqrt{\pi^d}}{\Gamma(d/2+1)} \cdot r^d$$

where $\Gamma(n)$ is the gamma function ($\Gamma(x+1) = x \cdot \Gamma(x), \Gamma(1) = 1$ and $\Gamma(1/2) = \sqrt{\pi}$). Since

$$sp^d(dist^{nn}) = \frac{1}{N}$$

we can get the expected nearest neighbor distance

$$dist^{nn}(N,d) = \frac{1}{\sqrt{\pi}} \cdot \sqrt[d]{\Gamma(d/2+1)N}$$

Based on this formula, we can determine that $dist^{nn}$ can become larger than 1 when the dimensionality is beyond 40. Because of the large NN distance, it is almost impossible to partition the data space well, thus tree-like index structures cannot be efficient for uniform datasets. The index-based search needs to access all the data when the dimension is more than 60. From the above analysis, we can see that sequential scan is a good choice when the dimensionality is high, because it avoids complicated tree operations, although it also has to compute the distance between query point and the whole datasets.

As discussed in the above cases, the expensive distance computations are inevitable. Thus, we want to use a fast distance filter to prune out most of the points which are less possible to be KNN before we access the real vectors, although we may not get the exact KNN. While the VA-file uses a filter step in its operations, it can reduce the disk accesses but leads to higher computational cost. A more efficient filter that can reduce the distance computational cost is needed in main memory systems.

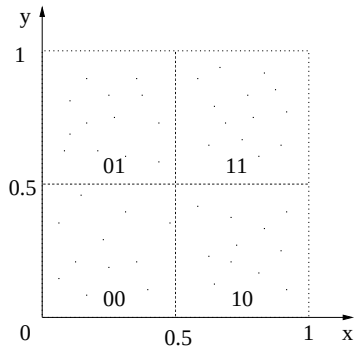


Fig. 1. Bits for 2 dimensional random data

Figure 1 shows the random data in a 2-dimensional space. We use one bit to represent each dimension, and the whole dataspace can be split into four partitions, represented by ”00”, ”01”, ”10” and ”11” respectively. In general, ”0” represents the vector value ”0-0.5” and ”1” represents the vector value ”0.5-1”. We call the mean value 0.5 the $bitcoder$. Suppose we have a 2-d point ”0.3, 0.1”, we can simply represent it by a bit-string ”00”. Using this method, we generate the bit-strings for all data points. Looking at the figure, we get the first impression that the points with same bit-string representation are generally nearby. Furthermore the partitions with less bit-difference are also nearer, e.g. partition ”00” is nearer to ”01” or ”10” than partition ”11”. With the bit-strings, we can filter the potential far points only using bit comparisons and avoid expensive distance computation. Of course, there exists some

exceptions, e.g. the points near the border. However, since our scheme is to find the approximate KNN, this trade-off between quality and processing time is promising if we can provide high approximation quality.

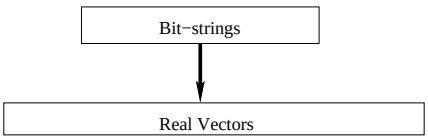


Fig. 2. The structure of BID

The structure of the BID is shown in Figure 2. The first level is the bit-strings of the real vectors, which is a flat directory. Each bit-string is related to a real data vector, all the bit-strings are stored in memory consecutively.

Figure 3 shows the algorithm to construct a BID structure. Since the first level of BID index is a flat directory with each record a bit-string, the construction algorithm is straightforward. For each real vector in the dataset, we compare the vector values with the $bitcoder$ 0.5 and determine the bit value in the bit-string B. The process stops when all the data points have been transformed into bit-strings.

Algorithm build_BID()
Input: Dataset
Output: BID index B

1. For i = 1 to N
2. For j = 1 to D
3. if (point(i)_j $\geq$ 0.5)
4. B(i)_j = 1;
5. else
6. B(i)_j = 0;
7. return BID index B;

Fig. 3. The algorithm of BID construction

Figure 4 shows the algorithm to search for approximate KNN using the BID. In the first stage, we initialize the approximate KNN candidate list and the pruning bit-difference, i.e the dimensionality D (lines 1-2). Next we transform the query point Q to a bit-string QB with length D (line 3). After that, we repeat the operations in lines 5-8 until the whole bit-string level of BID structure has been scanned. In this filter step, we compare two bit-strings, i.e. $B(i)$ and QB and get number of bits with different values. As we have discussed, the distance between two points is generally larger when they have more different bits. For each bit-string $B(i)$, we calculate the bit-difference from QB. If the bit-difference is less than the pruning bit-difference, we insert the i into the candidate list and update the $prune\_BID$ if necessary, which is always equal to the bit-difference between the query point and the K-th nearest neighbor candidate. Finally, we access real vectors of the dataset to get results.

The number of approximate KNN candidates selected in the filter step can affect the approximation quality. This number is determined by the pruning bit-difference. Suppose we need to find K approximate nearest neighbors, the bit-difference M is needed to pick at least K candidates in the filter step. However these candidates may not have sufficient

Algorithm AKNN()
Input: Dataset, BID index, Query point Q, K
Output: approximate K nearest neighbors

1. AKNN_C = Newlist();
2. prune_BID = D;
3. Transform Q to bit-string QB;
4. For i = 1 to N
5. Bit_dif = Get_BID(B(i), QB);
6. if (Bit_dif <= prune_BID)
7. insertion i into AKNN_C;
8. prune_BID = Bit_dif;
9. Access the dataset to get real vectors;
10. Return approximate K nearest neighbors;

Fig. 4. The algorithm of approximate KNN search with BID

approximation quality. Thus we can relax this constraint, e.g. using the bit-difference value $M + 2$. In other words, more candidates are retrieved after filtering. Finally, we provide K best answers after accessing the real dataset. The additional cost is more data access and distance computation of the real vectors. Clearly, the quality of approximation can be improved. We show more details in the experimental study.

B. The BID$^+$: an enhancement of the BID

Although the BID is expected to reduce the number of distance computations, it has several limitations. First, the BID inherently assumes that the data is uniformly distributed and uses mean value 0.5 to split each dimension. However, in real datasets, value distribution could be skewed. Second, each dimension has the same weight in BID, but some dimensions are more important in real applications. Third, real datasets are typically not globally distributed, and they may appear as clusters. We need to address these limitations to facilitate different kinds of high-dimensional datasets.

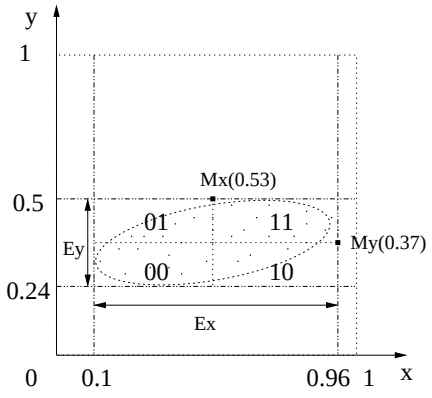


Fig. 5. Bits for skewed data

Figure 5 shows an example for skewed data. Unlike Figure 1, the data only occupies a small area of the data space. The data value extension is [0.1, 0.96] (E_x) on dimension x and [0.24, 0.5] (E_y) on dimension y respectively. Clearly on dimension y, the bit representation is 0 for all the point if we use 0.5 as a split criterion. The solution is that we use the mean value of extension as a $bitcoder$, e.g. M_x 0.53 and M_y 0.37 in the Figure 5.

On the other hand, the bit information of dimension x is more important than that of dimension y, as the dimension x has much large variance. Furthermore, even if two points have the same bit value on a certain dimension, it does not mean that the distance on this dimension is equal to 0. Therefore, we propose the square of average distance to set the bit weight since we apply the Euclidean distance metric. Now we present how to compute the weight.

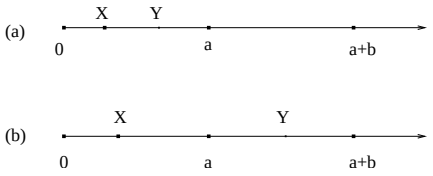


Fig. 6. An example for weight calculation

Suppose points X and Y have the same bit value, i.e. falling in [0, a] as shown in Figure 6 (a). Give a point X, the average distance between X and a random point Y, $Dist_0$, is

$$Dist_0 = \frac{\int_0^a |x-y|dy}{a} = \frac{\int_0^x (x-y)dy + \int_x^a (y-x)dy}{a} = \frac{a}{2} - x + \frac{x^2}{a} \quad (3)$$

The average distance between two random points, Ave_dist_0, is

$$Ave\_dist_0 = \frac{\int_0^a (\frac{a}{2} - x + \frac{x^2}{a})dx}{a} = \frac{a}{3} \quad (4)$$

Thus we can get the weight that two points have the same bit value, $Weight0$ is $(\frac{a}{3})^2$.

Suppose points X and Y have different bit values, i.e. falling in [0, a] and [a, a+b] respectively as shown in Figure 6 (b). Give a point X, the average distance between X and a random point Y, $Dist_1$, is

$$Dist_1 = \frac{\int_a^{a+b} (y-x)dy}{b} = a + \frac{b}{2} - x \quad (5)$$

The average distance between two random points, Ave_dist_1, is

$$Ave\_dist_1 = \frac{\int_0^a (a + \frac{b}{2} - x)dx}{a} = \frac{a+b}{2} \quad (6)$$

The weight that two points have the different bit value, $Weight1$, is $(\frac{a+b}{2})^2$. Note that both the weights are precomputed when we build the BID structure, so the weight calculation does not introduce any computational cost overhead during query processing. The sum of weights over the full dimensionalities, $\sum_1^D weight$, can be used to filter the points, where $weight$ can be $Weight0$ or $Weight1$ according to the bit comparison.

The BID scheme exploits the same $bitcoder$ to generate the bit-strings, however this method is not optimal as the real data could be clustered as shown in Figure 7. In this case the global bit-string representation will not yield good performance. As shown in [8], clustering is a good approach for approximate KNN query as it can reduce the retrieved set. The BID scheme benefits more from clustering as we can create $bitcoder$ for each cluster with respect to the cluster distribution, and hence the $bitcoder$ is much more efficient. In Figure 7, we use

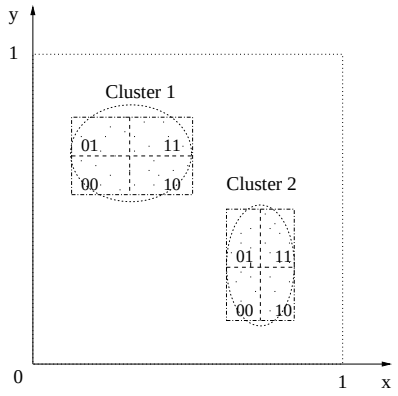


Fig. 7. Bits for clustered data

different $bitcoder$s for cluster1 and cluster2. In this paper, we employ the K-means clustering scheme to generate the clusters before we generate the bit-strings, and each cluster utilizes the cluster center as $bitcoder$.

We refer the BID scheme with the above enhancements as BID$^+$, and the structure of BID$^+$ is shown in Figure 8. The first level is a flat directory which store the cluster information. Each entry is a 4-tuple(c, r, w, ptr), where c is the cluster center and the coordinates of c are used as $bitcoder$, r is the radius of the cluster, w represents the dimensional weight which is a $2D$ array, and ptr is a pointer to the generated bit-strings.

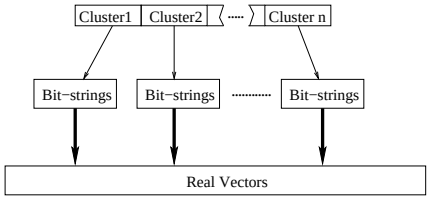


Fig. 8. The structure for BID$^+$

Algorithm build()
Input: Dataset
Output: BID$^+$ index B

```
1. Clustering dataset;
2. for each cluster
3.       init(U[D], L[D]);
4.       for each point P in the cluster
5.            for i = 1 to D
6.                 if (P_i > U_i)
7.                      U_i = P_i;
8.                 elseif (P_i < L_i)
9.                      L_i = P_i;
10.                if (P_i ≥ C_i)
11.                     b_i = 1;
12.                else
13.                     b_i = 0;
14.      Compute weights;
15. Return index B
```

Fig. 9. The algorithm of BID$^+$ construction

Figure 9 shows the algorithm for constructing a BID$^+$ structure. At first, we partition the dataset into clusters using the

K-means method. Meanwhile, we get the cluster information, such as cluster center, cluster size and radius. After that, we can process each cluster separately (lines 3-14). In Line 3, we initialize the upper bound $U[D]$ and lower bound $L[D]$ of the extension of each dimension, e.g. U_i = 1 and L_i = 0 where $i \in [1, D]$. For each point in the cluster, we have two tasks: first we update the upper and lower bound of the dimensional extension; second, we compare the vector values with $bitcoder$ (cluster center) and determine the bit value in the bit-string. Finally, we compute the weights for each dimension of cluster using the value of $U[D]$ and $L[D]$, and store the weights in the top level directory.

Algorithm AKNN$^+$()
Input: Dataset, BID$^+$ structure, query point Q, K
Output: Approximate K nearest neighbors

```
1. AKNN_C = Newlist();
2. prune_weight_sum = ∞;
3. prune_dist = ∞;
4. scan the cluster directory;
5. sort the cluster using dist(c, Q);
6. for each cluster
7.       if (dist(c, Q) - r) < prune_dist
8.            Transform Q to bit-string QB;
9.            for each bit-string
10.                weight_sum = 0;
11.                for i = 1 to D
12.                     if (Bit values are same)
13.                          weight_sum += Weight0_i;
14.                     else
15.                          weight_sum += Weight1_i;
16.                if (weight_sum <= prune_weight_sum)
17.                  insertion ID into AKNN_C;
18.                  prune_weight_sum = K-th weight_sum;
19.           prune_dist = the K-th exact distance;
20. Access the dataset to get real vectors;
21. Return approximate K nearest neighbors;
```

Fig. 10. The algorithm of approximate KNN search with BID$^+$

We summarize the approximate KNN algorithm of BID$^+$ in Figure 10. In the first step, we initialize approximate KNN candidate list, the pruning weight sum and pruning distance (lines 1-3), where the weight sum is used within the cluster but the pruning distance used to filter clusters. Next we scan the top level cluster directory, and sort the cluster by the distance between cluster centers and the query point. After that, we repeat the operations in lines 6-19 until all the qualified clusters have been examined. Note that the nearer clusters have the higher priority for processing. We transform the query point into bit-string using the cluster $bitcoder$ (line 8), then scan the bit-strings for bit comparison. For each dimension, if the point and query point Q have the same bit value, the $weight\_sum$ is added by $Weight0$; else the distance is added by $weight1$. If this approximate distance is less than the $prune\_weight\_sum$, we insert the record ID into the AKNN list and update the $prune\_weight\_sum$ if necessary, which is always equal to the $weight\_sum$ between the query point and the K-th nearest neighbor candidate. Once the cluster has been scanned, we compute the prune_dist by accessing the K-th real vector. This Distance is used to decide whether we need to continue processing other clusters. After all the qualified clusters have been checked, we access the real vectors to determine the approximate KNN answer. Similar to the BID, the $prune\_weight\_sum$ can be tuned to control the approximation quality of the BID$^+$. To improve the quality, we just relax this filtering constraint, e.g. using 120% of K-th weight_sum as a filter parameter. By retrieving more candidates and refining KNN after real vector accesses, we can provide the approximate KNN with higher quality.

IV. Performance Study

In this section, we present an experimental study to evaluate the proposed approximate KNN search structures, the BID and BID$^+$. The performance of each technique is measured by the average execution time, and the number of cache misses over 100 different approximate KNN queries. We use the $Perfmon$ tool [7] to count the L2 cache misses. All the experiments are conducted on a SUN E450 machine with 450 MHz CPU, 4 GB RAM and 2 MB L2 cache with 64 bytes block size. The machine runs SUN OS 5.7. All the data and index structures are loaded into the main memory before each experiment begins. We demonstrate the results on random dataset, synthetical clustered dataset and real-life dataset.

The experimental study includes two parts. First, we tune the performance of the BID and BID$^+$ and investigate the tradeoff between the execution time and the approximation quality. We apply two quality metrics as discussed in Section 2, $ratio\ of\ false\ dismissals$ (RFD) and $ratio\ of\ distance\ errors$ (RDE). Second, we compare them with some existing techniques for approximate KNN search in high-dimensional space. We use three reference techniques for the comparison: the approximate KNN-query evaluation technique based on the VA-file (VA-LOW) [19], the general framework for approximate KNN proposed in [8] ($Clustering$) and Sequential Scan. For clarity of presentation, we use some default parameters for these structures, e.g. 4 bits for the VA-LOW, and 10 clusters for the Clustering.

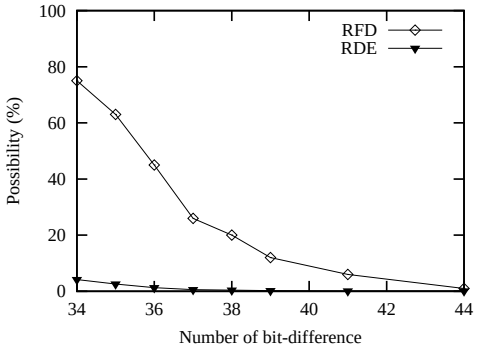
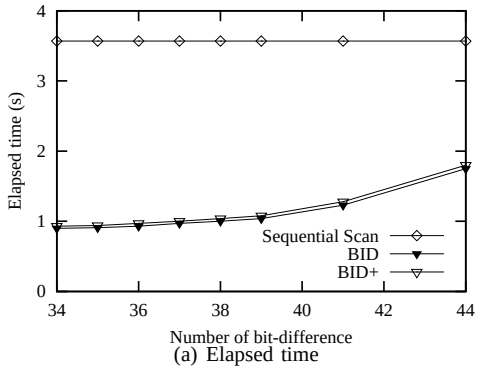


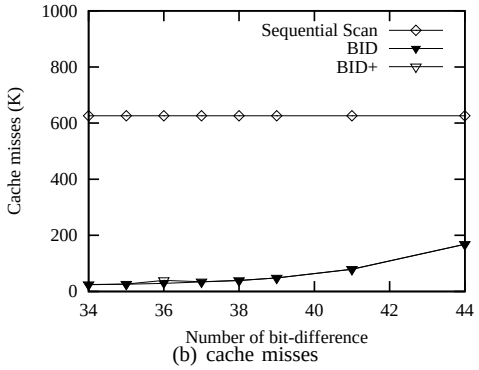
Fig. 11. The approximation quality for different bit-difference

A. Tuning the BID and BID$^+$

In this experiment, we first generate a uniformly distributed data set with 100 dimensions, the data size is 100,000. Since our methods use only one bit to represent each dimension, and hence there surely exists some information loss. However we can tune the bit-difference in the filtering step. If the

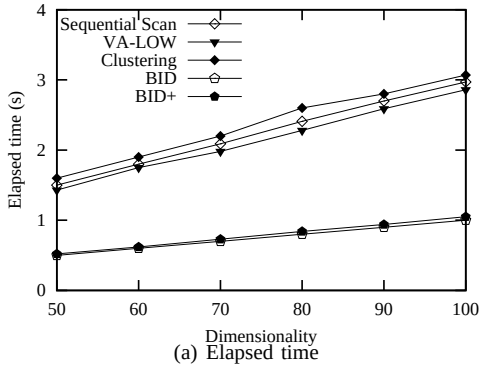


(a) Elapsed time

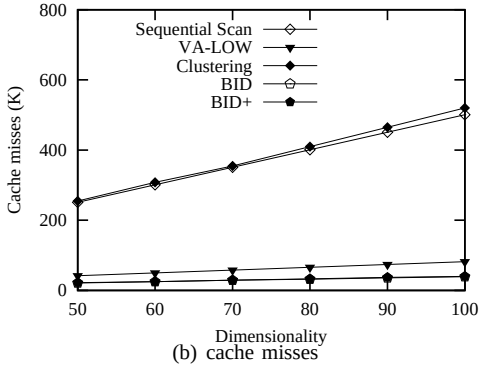


(b) cache misses

Fig. 12. The execution cost for different bit-difference



(a) Elapsed time



(b) cache misses

Fig. 13. The execution cost for different dimensionality

approximate quality is not sufficient, we can retrieve more candidates using a larger value of bit-difference, and finally finalize the answer after examining the exact distance.

Figure 11 shows the approximation quality by varying the numbers of bit-difference for 100-NN query. Note that all the dimensions have the same weights because of uniformly distributed data, hence the BID and BID$^+$ yields similar performance, and we only show the bit-difference as evaluation criterion for the clarity of presentation. The smallest value of bit-difference, 34, can guarantee finding 100 approximate nearest neighbors in the bit comparison step in this experiment. Although the $ratio\ of\ distance\ errors$ (RDE) is only 4%, the $ratio\ of\ false\ dismissals$ (RFD) is as high as 75% which means only 25% exact nearest neighbors are retrieved. Because we use one bit to represent the information of a feature vector (float), bit-difference does not mean the real distance between two points and too much information loss was incurred during the filtering. For example, 0.51 and 0.49 have the different bit value, but the real distance is very small, only 0.02. On the other hand, we observe that as the number of bit-difference which is used for filtering increases, the approximation quality can be improved significantly, especially RFD. The RFD value decreases to 20% and 10% when we use 38 and 40 bit-difference respectively. The larger bit-difference introduces more KNN candidates retrieval, and hence the higher execution cost. The results are shown in Figure 12.

Both the execution time and the cache misses increases when we use larger bit-difference value. We need to retrieve more real vectors for distance evaluation. The real vector accesses incur more cache misses and computational cost. These cost overhead are traded for higher approximation quality. The BID$^+$ performs a bit worse than the BID because of the computation of $weight\_sum$. We use the Sequential Scan as a reference. The results clearly demonstrate the superiority of the proposed schemes over the Sequential Scan: the BID and BID$^+$ are more than 100% better in all the cases, even when both RFD and RDE are approaching 0, e.g. the number of bit-difference is 44.

B. Comparing with other structures

In this section, we compare the proposed structures with some existing methods on different datasets, such as VA-LOW, Clustering and Sequential Scan. To ensure a fair comparison, we exploit partial dataset access to meet the same approximation quality criterion for VA-LOW and Sequential Scan. For example, the Sequential Scan access only a portion of the dataset and answer the query based on the portion that is read, e.g. access 80% of the dataset if RFD is set as 20%.

1) On uniformly distributed dataset: In this experiment, we first generate a uniformly distributed dataset with 50 to 100 dimensions. The data size is 100,000 points. We set the RFD and RDE at 20% for approximate 100-NN query, and the results are summarized in Figure 13.

The figure shows that our proposed BID and BID$^+$ yield the best performance for this dataset. The BID can filter most of the points with less possibility to be the nearest neighbors without expensive distance computation, where only bit operations are involved. Additionally, since we only use

one bit to represent each dimension, the space cost of bit representations is only $\frac{1}{32}$ of original data. The number of real vector accesses is limited by the filtering, and hence the BID can reduce the cache misses significantly. The BID$^+$ is a bit worse than the BID, because the structure of BID$^+$ is more complicated and the enhancement of BID$^+$ cannot benefit from the uniformly distributed data.

The Sequential Scan needs 200% more response time than the BID. It has to access the majority of the data and compute the distance. The VA-LOW is about 5% better than the Sequential Scan. In the filter step of KNN searching, we must scan the entire approximations, decode the approximated bit-string and calculate the lower bounds on the distance to the query point. Although the VA-LOW is also computation sensitive scheme, it incurs much less cache misses than the Sequential Scan as shown in Figure 13 (b), and hence the overall execution time is not so expensive. However the KNN search cost is bounded by the computational cost in main memory environment, thus the schemes based on VA-file are not optimal for memory processing. The Clustering method is worst among these methods, as it cannot partition uniform dataset efficiently and have to access most of the data.

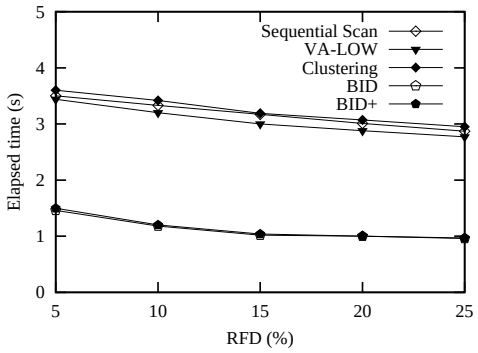


Fig. 14. The performance for different RFD

Figure 14 shows the approximate 100-NN performance for different RFD from 5% to 25%. The dimensionality is fixed as 100. Since all the methods can provide high quality RDE, we do not present the details here. When the RFD increases, all the methods yield better performance, as lower demand of approximation quality can reduce the data access and distance calculations. For the VA-LOW, it can use fewer bits representation, i.e. fewer cache misses. The BID and BID$^+$ are superior in all the cases because of their low computational cost and cache misses.

2) On skewed dataset: In many applications, data points are typically skewed in some ways. In this set of experiments, we evaluate these methods on skewed datasets with 100 dimensions. We use a method similar to that of [4] to generate the skewness in subspaces of different dimensionalities. The dataset has 100,000 points. The performance of BID$^+$ is affected by the number of clusters when the data is skewed or clustered, which is different to the random dataset. For the fair comparison, we adapt the same cluster number as the Clustering scheme throughout the experimental study.

Figure 15 shows the approximate 100-NN performance for different RFD from 5% to 25%. Clearly the performances are different from the uniform data. The Sequential Scan and VA-LOW perform worse than other methods, because it have to access most of the items and compute the distance regardless of data distribution. The BID degrades in this experiment because of lower filtering efficiency. Since the vector values of points are skewed, the BID cannot prune the point efficiently and introduce much more false hits, e.g. many points may have same bit value on a certain dimension. Therefore, the BID has to access more real data and compute the distance to determine the KNN, i.e. more cache misses and computational cost.

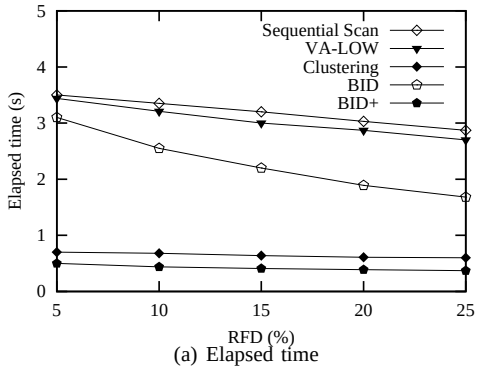
The BID$^+$ and Clustering provide much better results. The skewed dataset can be efficiently partitioned by clustering algorithm, thus these two methods only need to access some nearest clusters to get the approximate KNN. The BID$^+$ is about 35% better than the Clustering, because it can prune the majority data in the filtering step. Although the data is skewed, the enhanced coding mechanism and dimensional weight calculation can improve the filtering efficiency compared with BID.

3) On real dataset: In this experiment, we evaluate the various schemes on a real-life data, which contains 64-dimensional color histograms extracted from 70,100 images. We test the performance of KNN search with different approximation quality.

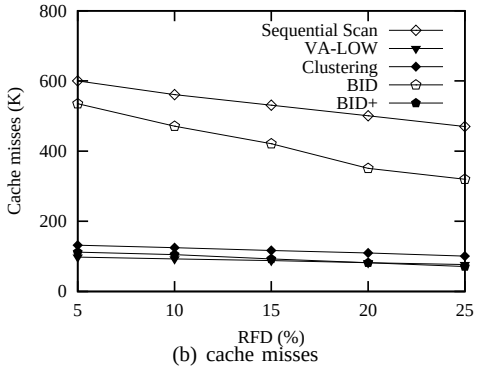
Figure 16 shows the approximate 100-NN performance by varying the value of RFD from 5% to 25%. These results are similar to the synthetic dataset in previous experiment and clearly show the effectiveness of the BID$^+$. The BID$^+$ is about 40% faster than the Clustering method and more than 200% better than other methods in terms of execution time cost. The real-life datasets are typically skewed and clustered, and hence the BID cannot prune the data efficiently during the filtering step. Because the real vectors are skewed, many bits may have same bit value when we transform the float to one bit, the bit-difference cannot distinguish the distance well. The BID$^+$ and Clustering are the best two choices for this dataset, as the color histogram dataset can be clustered and some nearest clusters can provide the enough approximate KNN. With the bit-difference comparison, the BID$^+$ can further decrease the computational cost and cache misses.

V. Conclusion

In this paper, we have revisited the problem of accessing high-dimensional data and develop a novel index structure to support efficient approximate KNN query in main memory environment. To reduce the distance computation, we first proposed a methodology using Bit-Difference (BID) to answer approximate KNN query. The BID employs bit-string to represent each point and the bit-difference is used to prune to the data points. To facilitate skewed dataset, we proposed an improved structure, named BID$^+$, which is enhanced with clustering, cluster adapted $bitcoder$ and dimensional weights. Extensive experiments are conducted to show that the BID$^+$ scheme yields significant performance advantages over the existing index structures on both real life and synthetic high-dimensional datasets.

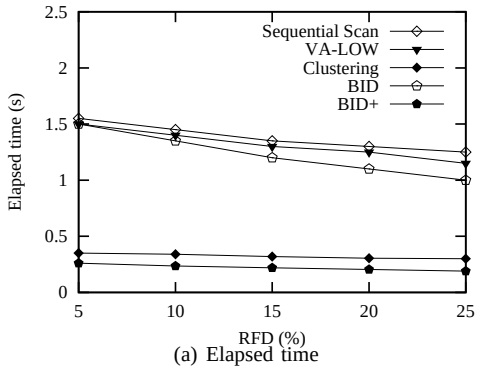


(a) Elapsed time

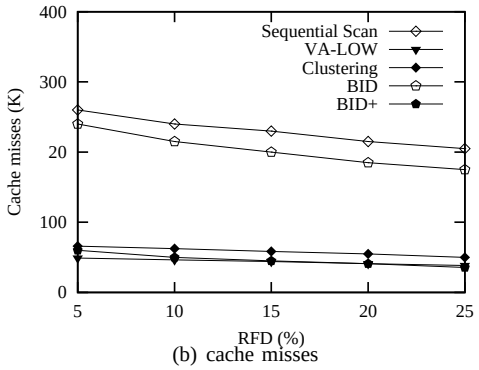


(b) cache misses

Fig. 15. The execution cost for RFD



(a) Elapsed time



(b) cache misses

Fig. 16. The execution cost for RFD

In future work, we want to investigate the effect of weight, cluster number and pruning constraint (bit-difference and weight_sum) over the approximation quality, including theoretical analysis and cost model. We also plan to adapt our technique into real life application such as indexing large image database.

REFERENCES

[1] S. Acharya, P.B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *Proc. of the ACM SIGMOD Conference*, 1999.

[2] S. Berchtold, C. Bohm, D. Keim, F. Krebs, and H. P. Kriegel. On optimizing nearest neighbor queries in high-dimensional data spaces. In *Proc. 8th ICDT Conference*, pages 435–449, 2001.

[3] C. Bohm, S. Berchtold, and D. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. In *ACM Computing Surveys*, pages 322–373, 2001.

[4] K. Chakrabarti and S. Mehrotra. Local dimensionality reduction: A new approach to indexing high dimensional spaces. In *Proc. 26th VLDB Conference*, pages 89–100, 2000.

[5] B. Cui, J. Hu, H. T. Shen, and C. Yu. Adaptive quantization of the high-dimensional data for efficient knn processing. In *Proc. 9th DASFAA Conference*, 2004.

[6] B. Cui, B. C. Ooi, J. W. Su, and K. L. Tan. Contorting high dimensional data for efficient main memory processing. In *Proc. of the ACM SIGMOD Conference*, pages 479–490, 2003.

[7] R. Enbody. *Perfmon: Performance Monitoring Tool*. available from http://www.cps.msu.edu/ enbody/perfmon.html, 1999.

[8] H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, and A. E. Abbadi. Approximate nearest neighbor searching in multimedia databases. In *Proc. 17th ICDE Conference*, 2001.

[9] R. F. S. Filho, A. Traina, C. Traina Jr., and C. Faloutsos. Similarity search without tears: the omni-family of all-purpose access methods. In *Proc. 17th ICDE Conference*, 2001.

[10] J. Goldstein and R. Ramakrishnan. Contrast plots and P-Sphere tree: Space vs. time in nearest neighbor searches. In *Proc. 26th VLDB Conference*, pages 429–440, 2000.

[11] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proc. 30th ACM STOC Conference*, pages 604–613, 1998.

[12] C. Traina Jr., A. Traina, C. Faloutsos, and B. Seeger. Fast indexing and visualization of metric data sets using slim-trees. *IEEE Transactions on Knowledge and Data Engineering*, 2002.

[13] K. Kim, S. K. Cha, and K. Kwon. Optimizing multidimensional index trees for main memory access. In *Proc. of the ACM SIGMOD Conference*, pages 139–150, 2001.

[14] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Approximate nearest neighbor searching in multimedia databases. In *Proc. 30th ACM STOC Conference*, 1998.

[15] V. Poosala and V. Ganti. Fast approximate answers to aggregate queries on a data cube. In *Proc. of the 11th SSDM Conference*, 1999.

[16] J. Rao and K. Ross. Making B+-trees cache conscious in main memory. In *Proc. of the ACM SIGMOD Conference*, pages 475–486, 2000.

[17] Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima. The A-tree: An index structure for high-dimensional spaces using relative approximation. In *Proc. 26th VLDB*, pages 516–526, 2000.

[18] J. S. Vitter, M. Wang, and B. Iyer. Data cube approximation and histograms via wavelets. In *Proc. of the 7th CIKM Conference*, 1998.

[19] R. Weber and Klemens Bohm. Trading quality for time with nearest-neighbor search. In *Proc. of the EDBT Conference*, 2000.

[20] R. Weber, H. J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. 24th VLDB Conference*, pages 194–205, 1998.

[21] C. Yu, B. C. Ooi, K. L. Tan, and H. V. Jagadish. Indexing the distance: An efficient method to KNN processing. In *Proc. 27th VLDB Conference*, pages 421–430, 2001.