Noname manuscript No.
(will be inserted by the editor)

SC-LSH: having both accuracy and efficiency for approximate nearest neighbor search in high-dimensional space

Feng Xiaokang · Cui Jiangtao · Li Hui · Liu Yingfan

Received: date / Accepted: date

Abstract The dimension curse and the I/O bottleneck are two major challenges for external Approximate Nearest Neighbor (ANN) search in high-dimensional space. Locality Sensitive Hashing (LSH) and its variants are among the most widely adopted solutions for this problem. However, most the state-of-the-art LSH-based methods incur a drawback: they cannot simultaneously achieve both high search accuracy and high I/O efficiency.

To address this issue, we recommend a novel method SC-LSH (SortingCodes-LSH) based on several simple yet effective measures. Firstly, we intensify a sorting-keys strategy by employing a better linear order to optimize the local distribution of the candidates as well as boosting the algorithm robustness. Then, we use compact binary codes instead of original data points to do candidate refining. These space-efficient data codes can enable us acquire significantly candidate codes via much less I/O operations. With the good similarity preserving ability, they are precise enough to discriminate nearest neighbors and thus guarantee the accuracy.

Empirical study on real-world datasets has demonstrated the superiority of SC-LSH in both the I/O efficiency and accuracy of ANN search compared with state-of-the-art LSH-based methods, including C2LSH, SK-LSH, SRS and QALSH.

Feng Xiaokang · Cui Jiangtao (✉)
School of Computer Science and Technology, Xidian University, China
E-mail: research@xkfeng.com
E-mail: cuijt@xidian.edu.cn

Li Hui.
School of Cyber Engineering, Xidian University, China
E-mail: hli@xidian.edu.cn

Liu Yingfan
Department of System Engineering and Engineering Management, Chinese University of Hong Kong, China
E-mail: liuyf@se.cuhk.edu.hk

Keywords Approximate Nearest Neighbor Search · Linear order · Locality Sensitive Hashing · Compact binary code

1 Introduction

Nearest Neighbor (NN) search in Euclidean space is a fundamental paradigm in various applications, such as information retrieval, machine learning, artificial intelligence, multimedia database, pattern recognition, data mining and so on. Due to the notorious "curse of dimensionality", the performance of most existing methods especially for exact nearest neighbor search decreases as the dimensionality increases and is even outperformed by the brute-force approach, linear-scan [3,46]. Besides, since most datasets in large applications are often too massive to fit into the main memory, disk-based (or external-based) NN search methods are required. However, most the existing methods require fetching a large set of candidates (typically hundreds or thousands) from the disk and thus incur significant disk I/O costs. The resulting I/O communication between fast internal memory and slower external memory sometimes become another major performance bottleneck of NN search [42].

To find an efficient solution, many researchers have been focusing on Approximate Nearest Neighbor (ANN) search recently, which aims at returning a point *close enough* to a query point instead of the closest one. Recently, Locality Sensitive Hashing (LSH) has been shown to be one of the most promising solution over ANN search. It employs distance-preserving hash functions to project nearby points into same bucket with high probability [15,21] which enables fast and accurate irrelevant points filtration with barely no pre-processing. Associated with the construction of compound hash functions and/or multiple hash tables, researchers can effectively reduce the false positives (FP) and false negatives (FN). Based on that, plenty of effective methods have been developed to further boost the ANN search performances, such as the accuracy (LSB-tree [40], C2LSH [13] and QALSH [19]), the I/O efficiency (SK-LSH [27]) and the space efficiency (SRS[38]).

However, a common limitation for most of the state-of-the-art LSH methods is that they cannot simultaneously achieve high accuracy as well as high efficiency (especially the I/O efficiency). Most of the methods that pursuit high search quality (such as C2LSH, QALSH and SRS) need to load and verify sufficient data objects which unavoidably consume enormous random I/O operations. Some others (SK-LSH [27], etc.) attempt to enhance the local distribution of the candidates so as to load more high-quality candidates during a single page access as well as improving the disk access manner (more sequential I/O, less random ones). However, they unfortunately endure poor scalability with high dimensions. Since the page size B is usually constant [1], as the dimensionality grows, less candidates can be held in a single disk page, which turns these methods ultimately resort to more disk page accesses to guarantee the search accuracy.

We devote to address the above issues to guarantee both a high search accuracy as well as a high I/O efficiency. Our method is composed of two simple yet effective measures.

Firstly, we take the sorting-keys strategy suggested in SK-LSH as the basic infrastructure of our method since it does work in improving the I/O performance of ANN search. However, we find that the linear order employed in SK-LSH [27] (i.e. the row-wise order) is not the best. Its "dimension-first-traverse"[2] property tends to introduce more false positives during the local rearrangement which makes it not able to adequately optimize the local distribution of the candidates. As a result, SK-LSH has to construct more hashing functions to boost the search accuracy. Besides, this linear order also makes SK-LSH very sensitive to the width W of the hashing functions.

We decide to improve the linear order for compound hash keys sorting in our method to intensify the local distribution of the candidates. We make a thorough investigation of how mechanisms of different space-filling curves affect the local distribution of NN candidates, and finally confirm a linear order [3] of "neighbor-first-traverse"[4] property, the Gray order suggested in [7] as the best choice for NN candidates distribution optimization. With that, we can further sharpen the sorting-keys strategy and also boost the algorithm robustness for ANN search.

Secondly, we no longer maintain any original data points in the index of SC-LSH. Instead, we store compact binary codes and load them to do candidate refining during the ANN search. All along, for the need of candidates refining, most LSH-based methods have to maintain one or more copies of the original data points in the index because the true distances between the query and the original data points are the most accurate reference to discriminate nearest neighbor. However, in most cases, they have become the major storage overhead of the index structure and also made the cost of loading and distance computing on candidate points cannot be underestimated. Besides, they are the immediate cause that limit the scalability of the I/O performance improving attempts in high dimensions as mentioned above. In a word, the original data is not the best choice in consideration of elevating both the accuracy and the I/O efficiency.

Actually, what we truly need in the candidate refining phase is not necessarily to be the original distance value but the ability of discriminating nearest neighbors. The compact binary coding technique is a very space efficient way that can also preserve the similarity among the original data points. The most promising work resides in the research of product quantization (PQ)-based techniques who manage to create sufficiently large amount of codewords to quantize the original data points based on the Cartesian product of divided subspaces, which gains significantly promotion in the quantization precision as well as high space efficiency of the data codes. Therefore, maintaining product quantization codes in our method could bring a great enlargement on the capacity of candidates of each disk pages which will enable us acquire significantly candidates via less I/O operations. With the good similarity preserving ability, they are precise enough to discriminate nearest neighbors during the candidate refining to ensure the search accuracy. Besides, this nearly constant space occupation of the data codes will also help us get rid of the limitation on I/O efficiency in high dimensions.

In summary, we introduce a novel method SortingCodes-LSH (SC-LSH for short) in this paper to simultaneously boost the accuracy and the efficiency (especially the I/O efficiency) of ANN search. Our main contributions are as follows:

[1] Note that the original SK-LSH method assumes the size B of a disk page as the number of data points contained in a disk page, we modify this setting to be more practical in this paper where B denotes the physical size of a disk page.

[2] Means that the curve will always traverse a higher priority dimension (or direction) first before entering into lower priority dimensions (or directions).

[3] Since the terms "linear order" and "space-filling curve" have similar meanings in our work, we will use them interchangeably in the following content.

[4] Means that the curve will always traverse a local neighborhood area first before entering into larger neighborhood areas.

- We make a thorough investigation on how mechanisms of different space-filling curves affect the local distribution of NN candidates and find that the "neighbor-first-traverse" curves can better improve the local distribution of NN candidates than "dimension-first-traverse" ones.
- Based on the above findings, we implement a sorting-keys strategy and choose the Gray curve with the "neighbor-first-traverse" property as the linear order of compound hash keys sorting to intensify the local distribution of the candidates so as to further reduce the I/O cost as well as enhancing the algorithm robustness of ANN search.
- We find that the crucial point in the candidate refining phase is the discriminating ability. Therefore, we introduce the space-efficient compact binary codes to replace original data points which can help us boost the I/O efficiency as well as retaining or even lifting the search accuracy since they are able to provide us with sufficient data codes with relative good similarity-preserving as candidates via much fewer I/O operations.
- We demonstrate the superiority of SC-LSH in both the I/O efficiency and accuracy of ANN search compared with state-of-the-art LSH-based methods (including C2LSH, SK-LSH, SRS and QALSH) by empirical study on real-life datasets.

The rest of this paper is organized as follows. Section 2 introduces several necessary preliminaries. Section 3 gives an overview of the technology road-map of SC-LSH. Detailed descriptions of SC-LSH are presented in Section 4 and Section 5 describes the experimental studies. Related work is discussed in Section 6. We conclude our work in Section 7.

2 Preliminaries

In this section, we list some necessary preliminaries about ANN search and LSH as well as the development of LSH-based methods that attempt to boost the accuracy and the efficiency of ANN search.

2.1 Problem definition

Given a dataset $\mathbf{D} \subset \mathbb{R}^d$ and a query point q, the target of NN problem is to find the point $o^* \in \mathbf{D}$ satisfying that for any point $p \in \mathbf{D}$, $\|o^*, q\| \le \|p, q\|$, where $\|\cdot,\cdot\|$ denotes the Euclidean distance between two points. In this paper, we focus on c-ANN, the popular approximate version of NN problem, which aims at finding a point $o \in \mathbf{D}$ that satisfies $\|o, q\| \le c\|o^*, q\|$. Here, c is the approximation ratio, which exceeds 1.

2.2 Locality Sensitive Hashing

2.2.1 Definition

To solve the c-ANN problem, Indyk and Motwani proposed the idea of LSH [21], which can ensure that a close pair collides with each other with a high probability and a far pair with a low probability. This property of LSH is also called distance-preserving. Here is a formal definition.

Definition 1 (**Locality Sensitive Hashing**) Given a distance R, an approximate ratio c and two probability values $\mathbb{P}_1$ and $\mathbb{P}_2$, a hash function $h : \mathbb{R}^d \to \mathbb{Z}$ is called $(R, c, \mathbb{P}_1, \mathbb{P}_2)$-**sensitive** if it satisfies the following conditions simultaneously for any two points $o_1, o_2 \in \mathbf{D}$:

- If $\|o_1, o_2\| \le R$, then $Pr[h(o_1) = h(o_2)] \ge \mathbb{P}_1$;
- If $\|o_1, o_2\| \ge cR$, then $Pr[h(o_1) = h(o_2)] \le \mathbb{P}_2$;

To make sense, both $c > 1$ and $\mathbb{P}_1 \ge \mathbb{P}_2$ hold.

Equation 1 shows a commonly used LSH function in Euclidean Space, which was proposed by Datar et al. [5].

$$h(o) = \lfloor \frac{a \cdot o + b}{W} \rfloor \tag{1}$$

Here, o is an arbitrary point in $\mathbf{D}$. a is a random vector with each dimension independently chosen from a Gaussian distribution. W is a real number representing the internal width of the LSH function and b is a real number uniformly drawn from the range [0,W). For two points o_1, o_2 and an LSH function h, if $\|o_1, o_2\| = r$, the probability of $h(o_1) = h(o_2)$ can be computed as follows [5].

$$\begin{aligned} p(r, W) &= Pr[h(o_1) = h(o_2)] \\ &= \int_0^W \tfrac{1}{r} f_2(\tfrac{t}{r})(1 - \tfrac{t}{W})dt \\ &= 2norm(W/r) - 1 - \tfrac{2}{\sqrt{2\pi}} \tfrac{r}{W}(1 - e^{-\frac{W^2}{2r^2}}) \end{aligned} \tag{2}$$

Here, $f_2(x) = \frac{2}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$ and $norm(\cdot)$ represents the cumulative distribution function of a random variable following Gaussian Distribution. According to Equation 2, the probability $p(r, W)$ decreases monotonically when r increases but grows monotonically when W rises.

2.2.2 Candidate mechanism

Due to the distance-preserving property of LSH, data points that are distant from each other are less likely to collide and thus be filtered out, while closer pairs are retained with high probability and become NN candidates. Based on this idea, several approaches have been proposed for c-ANN [15,5,29,40,13]. However, it is obvious that Equation 1 exhibits poor performance in filtering irrelevant points, as many distant pairs could also collide with each other under a single hash function and thus bring numerous false positives. To remove the false positives, usually a **compound LSH function** $\mathcal{G} = (h_1, h_2, \ldots, h_m)$ is employed so as to improve the distinguishing ability.Here each element of a compound LSH function, h_i, is randomly generated according to Equation 1. Specifically, for $\forall o \in \mathbf{D}$, $K = \mathcal{G}(o) = (h_1(o), \ldots, h_m(o))$ is called the **compound hash key** of point o under $\mathcal{G}$.

The basic LSH [5] suggests only points sharing all the m hash values of the compound hash key with the query point should be taken into account as candidate points. This is too strict to return enough number of result points, as a result, multiple compound hash functions have to built to make up with each other. Originally, hundreds or thousands compound hash functions are constructed, since one compound hash function corresponds to one hash table, the total storage consumes an extremely high amount of space which seriously affect the scalability and usage in high-dimensional data space.

To address this problem, more c-ANN search algorithms begin to identify candidates by evaluating similarity over compound hash keys. Either they directly probe the neighbor buckets, such as Entropy-LSH, multi-probe LSH, etc., or define a distance measurement over compound hash keys, such as LSB [40], C2LSH [13], SK-LSH [27], SRS [38], QALSH [19], etc. However, most of these methods collect their candidates among a large number of hash structures, and the candidates are distributed among different disk pages. Thus, a large number of I/Os are unavoidable to obtain sufficient candidates to guarantee the satisfactory accuracy of the returned results.

2.3 Sorting-keys strategy

To improve the I/O efficiency, there are mainly two measures, 1) reduce unnecessary I/O operations, and 2) improve the I/O manner as well as the hit rate of high-quality candidates during each page access. Recently, Liu et al. proposed a sorting-keys strategy SK-LSH which can achieve both the above goals.

SK-LSH introduces linear orders into compound hash keys to do candidate rearrangement. Linear orders are defined by space-filling curves. The latter is a kind of virtual curve manually constructed in the multidimensional space which can project the multi-dimensional space into one-dimensional integer keys. The space is firstly divided into equal-sized grids, each grid is given a unique id, the ascending order of the ids is the linear order stipulated by the curve. An important advantage of space-filling curves is that they are practically insensitive to the number of dimensions if the one-dimensional keys can be arbitrarily large. Everything is mapped into one-dimensional space which makes a lot of one-dimensional access methods, such as B tree, B$^+$-tree can be applied to manage the data [12].

Common used space-filling curves include the row-wise curve, the z-ordering, the Gray curve and the Hilbert curve [12]. Fig.1 shows some kinds of them. Generally, all these curves share a common property: neighbor grids along the curves are more likely to be neighbors in the original space. This is called the “clustering property” of space-filling curves. If one stores the data objects on the disk according to their linear orders, it is more likely to load NN candidates via sequential disk operations.

In the sorting-keys strategy, SK-LSH chooses to rearrange the compound hash keys under the row-wise curve to optimize the local distribution of candidates, because the compound LSH keys can well preserve the similarity among original data points. As a result, neighboring candidates are stored on the same or adjacent disk pages. It then defines a distance measure between compound hash keys to estimate the true distance between data points. During ANN search, a limited number of pages on the disk, which are “close” to the query in terms of the distance defined between compound hash keys, are needed to be accessed for sufficient candidate generation, leading to much shorter response time due to the reduction of random I/O operations, yet with higher search accuracy.

3 Overview of our method

In this section, we give deep explanations of the two major technical contributions in SC-LSH, the seeking for better linear orders and the reason we employ compact binary codes.

3.1 Sorting keys under better linear order

We take the sorting-keys strategy as the basic infrastructure of SC-LSH because it does help to improve the I/O performance, including both the I/O amount reduction and the I/O manner improving. However, as for the choice of linear order, we find that there are different kinds of space linear orders with different properties, and these properties have different effects on the local distribution of candidates.

3.1.1 Mechanisms of space-filling curves

Generally speaking, we can divide the space-filling curves into two categories according to their differences in setting the priority of space filling, the *“dimension-first-traverse”* curve (short for DFT) and the *“neighbor-first-traverse”* curve (short for NFT). Fig. 1 lists several samples of them.

Fig. 1(a) depicts a kind of DFT curve, the row-wise curve, which is the curve employed in SK-LSH. Fig. 1(c)(d)(e) depicts three kinds of curves, the z-ordering, the Gray curve and the Hilbert curve, respectively.

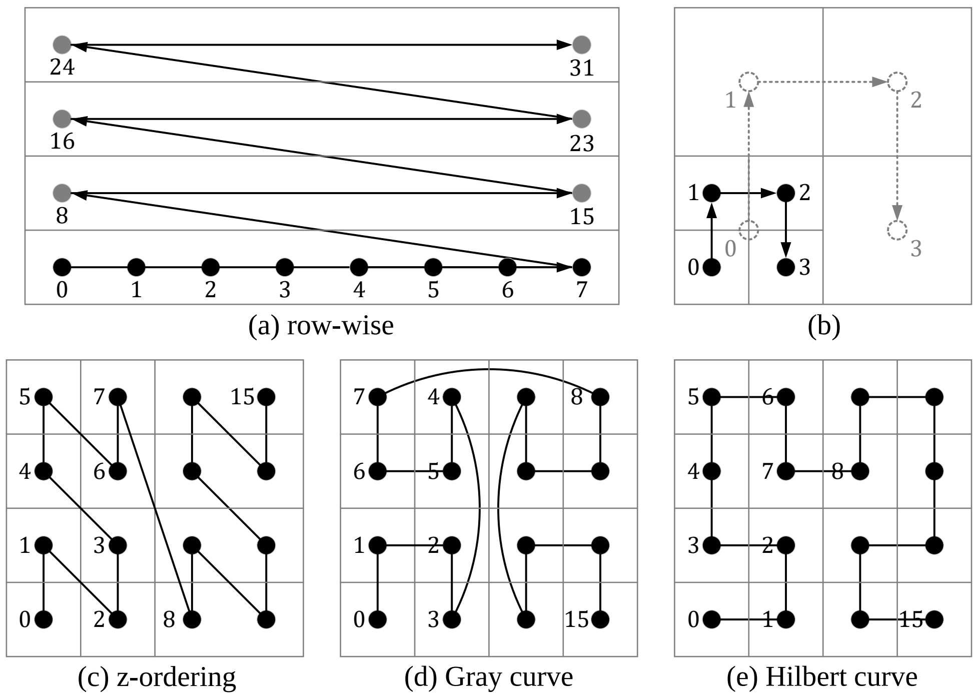


Fig. 1: Different kinds of space-filling curves

We can see clearly the differences on the priority they set for which to first traverse between these two categories of curves. For the row-wise curve, it specifies different priorities among different dimensions and will first traverse dimensions with higher priorities. Only when it goes through all the grids along the higher priority dimension, will it turn to the sub high-priority dimension to shift one step forward. After that, it turns back to the higher priority dimension and continues the filling. Suppose (x, y) denotes the coordinate of a two-dimensional grid, and the X dimension has higher priority than the Y dimension. Its order on the curve will be $(y \cdot dim + x)$, where dim represents the total number of coordinates along the X dimension.

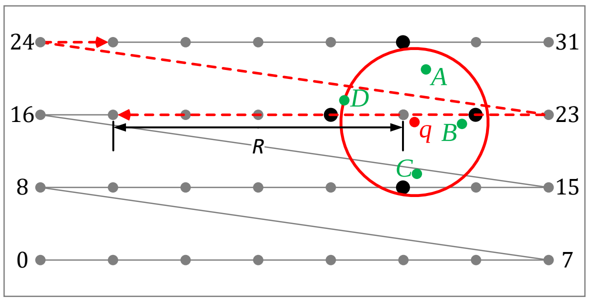
NFT curves will first fill up a small local space, after that, they expand to a larger neighbor area and continue the filling. Usually, it exhibits an obvious recursion trend of a pre-defined filling logic during the expansion. Taking the Gray curve in Fig. 1(d) as an example, the filling order conducted in the lower left 2×2 grids is also the order to fill up the larger space when zooming in the whole 4×4 grids as a bigger 2×2 grids space as shown in Fig. 1(b). Similar trends are also exhibited in the other two curves in Fig. 1(c) and Fig. 1(e).

3.1.2 Better linear order

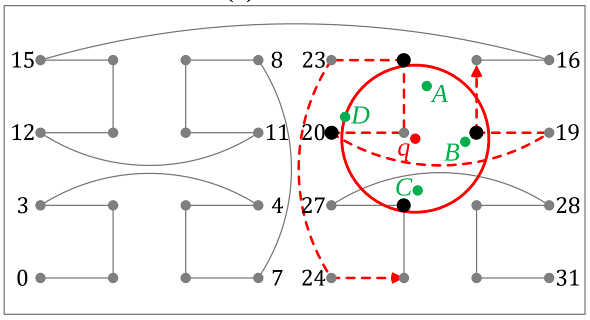
Different mechanisms of space linear order could have different effects on the distribution of the candidates. SK-LSH does not conduct further study on that and chooses the row-wise curve as the order for keys sorting, which in our opinion is not the best choice. Better linear order should be able to better optimize the local distribution of NN candidates so that the candidates can be loaded in fewer I/O operations during the NN search. It is intuitively that the NFT orders could produce better NN candidates distribution than the DFT ones since NFT orders are more in line with the principle of nearest neighbor seeking for the nearest neighbors are usually distributed around the query point.

To verify our judgment, we conduct a series of quantitative experiments. We generate a set of synthetic datasets whose cardinality are fixed at 200,000 and the dimensionality vary in $\{2, 4, 8, 12, 16\}$ (that can cover the dimensionality we need to conduct the linear order in our method in the following parts). For each dataset, the coordinates are randomly generated following a uniform distribution under the range of $[0, S]$, S can be seen as the scale of the coordinates in each dimension. In our generation, we set $S = 1{,}024$ for ease of parameter controlling.

For each dataset, we first split the whole space into equal-sized grids of width W. Then a certain kind of space-filling curve is conducted on all the grids. In this way, each data point can get its order alongside the curve according to the grid it locates in. We store all the data points on the disk according to their linear orders. When a query q arrives, we first specify a search range R. Then we start a bidirectional search from the grids where q locates in and load at most R grids respectively in each direction. We count how many true k-NNs of q can be found in the loaded data points. The



(a) row-wise



(b) G-order

Fig. 2: NN search test along two kinds of linear orders

recall rate *recall@k* of the true k-NNs is used to judge the quality of the NN candidates distribution under a certain linear order.

To make a clear illustration, we take the NN search case in Fig. 2 as an example. Fig. 2 depicts two NN search cases in the 2-dimensional space. The grid width W is set as 128, therefore, each dimension we can divide at most $n_I = S/W = 8$ intervals (the edge of grid) as shown. We conduct two linear orders (the row-wise curve and the Gray curve) respectively to represent the two categories of linear orders. Suppose q is the query point, $\{A, B, C, D\}$ are its true 4-NNs. We set the search range $R = n_I/2$ which means $R = 4$. Therefore all the grids that can be loaded will be determined (i.e., all the grids traversed by the red dotted lines in Fig. 2). As can be seen, on the row-wise order, we can get 2 true NNs, $\{B, D\}$, therefore the recall $recall_{row}@4 = 50\%$. As for the Gray curve, we will get three true NNs, $\{A, B, D\}$, the recall rate $recall_{Gray}@4 = 75\%$.

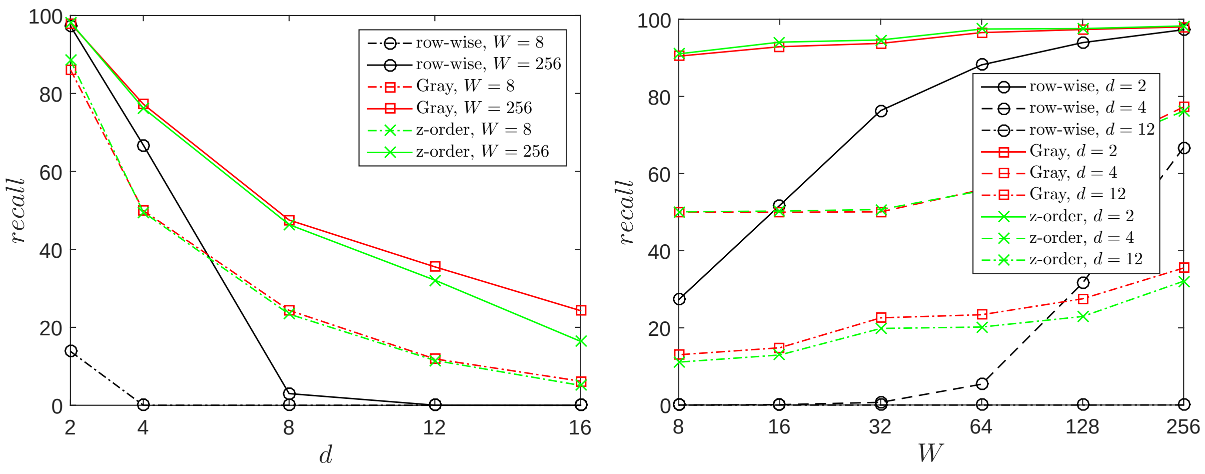
Similar to the above settings, we conduct the NN search comparison between the DFT orders and the NFT orders on the generated datasets. We choose the row-wise curve from the DFT category and the Gray curve as well as the z-order curve from the NFT category. Two parameters are tuned respectively to conduct different comparisons, one is the dimensionality d, as shown in Fig. 3(a), the other is the grid width W, as shown in Fig. 3(b). As for the recall target, we set $k = 100$ to investigate the recall rate of the query's true 100-NNs.

All the comparison results are listed in Fig. 3. As shown, it is obvious that the row-wise curve is more sensitive to both the data dimensionality d and the grid width W than the other two curves.

In Fig. 3(a), the recall rate on the row-wise curve exhibits a sharper decreasing trend with the growth of d. When d is larger than 8, the NN search recall in both grid widths are nearly 0. This indicates that the row-wise curve is very easy to loss nearest neighbors in higher dimensions. This is caused by its dimension-first-traverse property, which is also the reason why point A, C are missed in Fig. 2: because A and C are not located on higher priority dimension of the row-wise curve, they have to wait for one complete traverse along the other dimension before being accessed. And since most dimensions are of relatively lower priority, this could be more often when the dimension increases. As for the other two curves, the neighbor-first-traverse property enables them traverse more NN candidates earlier in the local neighborhood than the DFT ones which makes them bear less influence from the data dimensionality. Although we cannot conduct linear orders on arbitrary high dimensions according to Fig. 3(a), for the dimensionality we need in our method (around 10-20), the NN distribution quality is good enough to support effective ANN search coupled with dimension reduction techniques with LSH.

We also set the linear order comparison versus different W because we hear that SK-LSH is somewhat sensitive to the internal width W. If that is true, it may become another flaw of SK-LSH which could degrade the robustness of the ANN search algorithm. To verify that, we vary the grid width W in a relatively wide range, from the relatively small width $W = 8$ to a relatively large width $W = 256$ (note that the scale S of each dimension is at most 1,024) as shown in Fig. 3(b). Given a certain grid width W, the search range can be determined as $R = n_I/2 = \frac{S}{2W}$.

The result confirms our judgment. As can be seen in Fig. 3(b), when the width narrows, the recalls on all three curves exhibit decreasing trend. On the same dimension, the row-wise curve owns the most significant decreasing trend among them three. Again, this is caused by the characteristic of the curve, because the dimension-first-traverse property leads the row-wise curve to capture only a limited part of the real neighbors in the local neighborhood, much of the rest are missed. To ensure the query accuracy, one can enlarge the interval width so as to *catch* those missed neighbors *back* earlier. That is the reason why SK-LSH prefers to larger internal widths. However, the question is, enlarging the interval width W will increase the collision probability of two distant points according to Equation 2, which means more false positives will be introduced into the candidate set and thus reduce the query accuracy. To make up of that, more LSH functions have to built to filter out these false



(a) *recall* versus *d* (b) *recall* versus *W*

Fig. 3: Linear order comparison on synthetic datasets

positives adequately, which ultimately increases the index size as well as the computation overhead.

As for the NFT curves, they look not so sensitive to the change of W as shown in Fig. 3(b). Although they also exhibit increasing trend with the help of enlarging W, they are not very dependent on it since they can produce relatively good recall of nearest neighbors even on narrow internals.

In conclusion, different kinds of space-filling curves do have different effects on the local rearrangement of NN candidates, and the NFT curves perform better than DFT curves. This means NFT curves can help to further reduce the amount of I/O operations during the ANN search. Therefore, we will choose a kind of NFT curve for compound hash keys sorting in the following construction of SC-LSH.

As for the choice, we possibly have three options, the z-ordering, the Gray curve and the Hilbert curve. Actually, the Hilbert curve can achieve the best clustering property to preserve the locality of data points among them three as reported in [8], however, it is a pity that the Hilbert curve is somewhat difficult to implement in the dimensions of the compound hash keys we need in SC-LSH. The Gray curve performs better NN distribution than the z-ordering curve as indicated in Fig. 3 and is as easy to implement as the z-ordering. Therefore, we finally choose the Gray curve.

3.2 Candidate refining with compact binary codes

3.2.1 Limitations of maintaining original data

Most LSH-based variants need to maintain one or more, sometimes hundreds or thousands, copies of original data points due to the need of discriminating real NNs from the candidate set in the refining phase. However, this sometimes becomes another typical limitation of LSH based methods, which concretely speaking, mainly resides in two aspects.

- **Expensive space consumption** The space complexity of one copy of the original data points is $O(nd)$, for large scale high-dimensional dataset, this is very expensive and sometimes becomes the major space consumption of the index.
- **Degrading the I/O scalability** The maintenance of original data points will degrade the I/O scalability in terms of dimensionality of some methods devoting for I/O efficiency improving. Suppose that a dataset $\mathbf{D} \subset R^d$ resides in the external memory where each page occupies B *word*s. Then each page can hold nearly B/d data points. To guarantee the ANN search accuracy, we often need to load sufficient candidate points. Without loss of generality, suppose that we load γ points for each query, then the total number of disk pages we need to access is nearly $N_P = \gamma d/B = O(d)$ since γ and B can be seen as constants, which means the I/O cost is linearly proportional to the dimensionality. That is a poor scalability for external indexings.

The question is, do we necessarily have to keep the original data points in the index structure? The answer resides in the usage of them. We keep real data points because they can provide us with accurate distances between the candidates points and the query, and these distances can provide us with references to judge the quality the candidates so as to find out the real nearest neighbors. Hence, in fact, the actual point we care about is not the original distance value, but the ability of discriminating nearest neighbors. In other words, if we can find a space efficient way that can also preserve the similarity among real data points, we can avoid storing the original data and thus address the above limitations. Fortunately, the compact binary coding techniques can fulfill both the above requirements.

3.2.2 Effect of compact binary codes

In recent years, the computer vision community has witnessed rapid progress in learning similarity-preserving binary codes to encode high-dimensional data points into compact binary codes [16,41,47]. These codes could bring large efficiency gains in storage while preserving the similarity among the original data points. The most promising work resides in the research of product quantization (PQ)-based approaches [23,2,32,44,26].

As a quantization technique, PQ is derived from vector quantization (VQ), which reduces the representation space to a codebook $\mathbf{C}$ composed of $\mathbb{K}$ codewords (or centroids) $\{c_i\}_{=1}^{\mathbb{K}}$ generated from clustering algorithms. A data point $p \in \mathbf{D}$ is mapped into its nearest centroid under the Lloyd optimality conditions in terms of Euclidean distance:

$$\pi(p) = \arg\min_{c_i \in \mathbf{C}} \|p, c_i\|_2 \tag{3}$$

Here, $\pi(\cdot)$ is the quantizer, $\|,\|_2$ indicates the Euclidean distance and the index i can be seen as the quantized code of p. For any two points $p_i, p_j \in \mathbf{D}$, their distance can be estimated according to the quantized codes, i.e., $\left\|p_i, p_j\right\|_2 \approx \left\|\pi(p_i), \pi(p_j)\right\|_2$. For vector quantization, to obtain precise estimated distances, the quantization error must be limited. Therefore, the total number $\mathcal{K}$ of centroids should be sufficiently large. However, for typical vector quantization techniques, such as K-Means and hierarchical K-Means (HKM), one issue is the scalability of codebook. It is hard for them to scale to large $\mathcal{K}$ (e.g. $\mathcal{K} = 2^{64}$) since both the codebook storage and data point assignments become untenable.

PQ successfully addresses this issue with a model of compositional parameterization of cluster centroids. Suppose that d is a multiple of $\mathcal{M}$ and $\mathcal{S} = d/\mathcal{M}$. In PQ-based methods, the data space is firstly decomposed into a Cartesian product of $\mathcal{M}$ $\mathcal{S}$-dimensional subspaces, and each vector $p \in \mathbf{D}$ can be seen as a concatenation of $\mathcal{M}$ disjoint $\mathcal{S}$-dimensional sub vectors $u_j(p)$, $1 \leq j \leq \mathcal{M}$:

$$p = \underbrace{p^1, \cdots, p^{\mathcal{S}}}_{u_1(p)}, \cdots, \underbrace{p^{d-\mathcal{S}+1}, \cdots, p^d}_{u_{\mathcal{M}}(p)}$$

Then, K-Means is performed to obtain a sub codebook containing k^* sub codewords in each subspace. By this means, a codebook $\mathbf{C}$ containing $k^{*\mathcal{M}}$ centroids is generated using the Cartesian product of these sub codebooks:

$$\mathbf{C} = \mathbf{C}_1 \times \cdots \times \mathbf{C}_{\mathcal{M}}$$

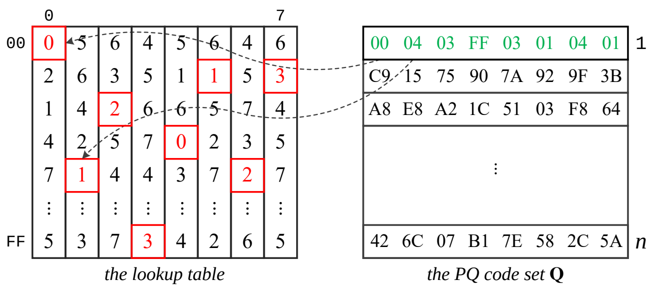


Fig. 4: Illustration of distance estimation using a lookup table

Note that the storage of the codebook $\mathbf{C}$ is only $O(k^* d)$, while K-Means requires $O(k^{*\mathcal{M}} d)$ storage with the same number of centroids. Using these sub codebooks, a data point p can be mapped as follows:

$$\begin{aligned} p \rightarrow \pi(p) &= (\pi_1(u_1(p)), \ldots, \pi_{\mathcal{M}}(u_{\mathcal{M}}(p))) \\ &= (\mathbf{C}_1[i_1], \ldots, \mathbf{C}_{\mathcal{M}}[i_{\mathcal{M}}]) \end{aligned}$$

Here, $\pi_j(\cdot)$ is the subquantizer in the j-th subspace. By concatenating the indexes of the $\mathcal{M}$ centroids returned by the $\mathcal{M}$ subquantizers, we can encode the data point p into a compact PQ code:

$$Q(p) = (i_1, \ldots, i_{\mathcal{M}})$$

Note that the time complexity of the data point encoding is reduced from $O(k^{*\mathcal{M}} d)$ in K-Means to $O(k^* d)$ here. Besides, k^* is usually set to be a power of 2. Let $\mathcal{U} = \log_2 k^*$, each PQ code is thus a $\mathcal{U}\mathcal{M}$-bit binary string which will occupy $s = \mathcal{U}\mathcal{M}/32$ machine words in total.

In addition to the high estimation precision, PQ conducts a kind of *asymmetric quantizer distance (AQD)* to estimate the square distance between a data point p and the query vector q without quantizing q:

$$\|q, p\|_2^2 \approx \|q, \pi(p)\|_2^2 = \sum_{j=1}^{\mathcal{M}} \left\|u_j(q), C_j\left[i_j\right]\right\|_2^2 \tag{4}$$

This can be done efficiently with the help of a pre-computed lookup table. According to Equation 4, by pre-computing all the square distances between each sub query vectors $u_j(q)$ and the sub codewords in the j-th sub codebook $\mathbf{C}_j$, the AQD square distance computation between p and q is transformed into a table lookuping with the PQ code components in $Q(p)$ as the indexes. Inspired by [1], we depict this procedure vividly in Figure 4, where we generate 64-bit PQ codes with $\mathcal{M} = 8$ and $k^* = 256$.

Since the compact binary codes, especially the PQ series are very space efficient and are accurate enough to discriminate nearest neighbors due to their high quantization accuracy, we decide to store compact binary codes instead of the original data points to do candidates refining in SC-LSH. More specifically, we focus on product quantizers with 2^{64} centroids, which means each PQ code occupies 64-bit storage. According to the performance estimation conducted in [23] and [32], the 64-bit long codes are good enough to produce sufficiently high quantization accuracy for most typical high-dimensional datasets, especially for the datasets chosen in our experiments in Section 5. 64 bits means 2 words. This constant storage means a compression ratio of $d/2$ (suppose that each component of the data point is a one word long integer/float) of the storage of the original data set. Besides, it also means that one disk page (B words) now can hold $B/2$ data codes, not only gets rid of the limitation from the dimensionality d, but also be enlarged to a big constant. Therefore, a few page accesses are much enough to load sufficient candidates. Both the two limitations caused by maintaining original data points (mentioned at the beginning of this section) will be eliminated.

4 SC-LSH

In this section, we introduce all necessary details to apply SC-LSH to address ANN search, including the linear order deploying, the indexing construction and the ANN search algorithm.

4.1 G-order over compound hash keys

We choose the Gray curve as the linear order (denoted as G-order) conducted over the compound hash keys in SC-LSH. We firstly make some necessary definitions.

Definition 2 (**G value**) The Gray curve projects the data in the I^m space to an integer in I, denoted as $G : I^m \to I$. For a data point $o \in \mathbf{D} \subset R^d$, let $K = \{k_i\}_{=1}^{m} = \mathcal{G}(o), k_i \in I$ denotes the compound hash key of o under a compound hash function $\mathcal{G}$ which consists of m LSH functions. $G(o)$ is called the *G value* of o. According to [8], the computing of G value mainly consists of three steps, all of which are easy to implement.

Definition 3 (**Prefix of G values**) Given a data point $o \in R^d$ and its G value $G(o) = g_1 g_2 \cdots g_S$ where $S = um$ is the length of the G value, let l denote the length of the prefix, we define the prefix of the G value as:

$$pref(G,l) = (g_1 g_2 \cdots g_l) \quad (5)$$

Definition 4 (**Length of the non-common prefix of G values**) Given two G values $G_1 = (g_{1,1} g_{1,2} \cdots g_{1,S})$ and $G_2 = (g_{2,1} g_{2,2} \cdots g_{2,S})$, if $pref(G_1,l) = pref(G_2,l)$ and $pref(G_1,l+1) \neq pref(G_2,l+1)$, then we can define the length of the non-common prefix between G_1, G_2 as:

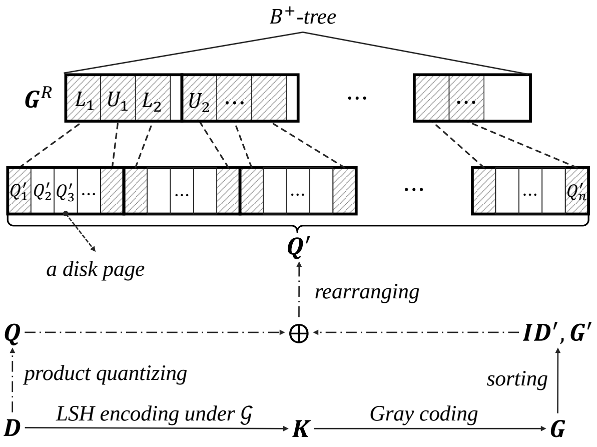


Fig. 5: Index construction

$$NL(G_1,G_2) = S - l \quad (6)$$

Specially, if $pref(G_1,S) = pref(G_2,S)$, we have $NL(G_1,G_2) = 0$.

Definition 5 (**Distance over G values**) For two data points $o_1, o_2 \in \mathbf{D}$, we define the distance between their G values G_1, G_2 as follows:

$$dist_G(G_1,G_2) = NL(G_1,G_2) \quad (7)$$

Definition 6 (**Partial order over G values**) For two data points $o_1, o_2 \in \mathbf{D}$, we define a partial order $\langle G, \leq_G \rangle$ for their G values G_1, G_2:

$$\begin{cases} G_1 <_G G_2, \; if \; l < S \; and \; g_{1,l+1} < g_{2,l+1} \\ G_1 =_G G_2, \; if \; l = S \\ G_2 <_G G_1, \; if \; l < S \; and \; g_{1,l+1} > g_{2,l+1} \end{cases} \quad (8)$$

here, $l = S - NL(G_1,G_2)$.

4.2 Indexing

In SC-LSH, all the data are deposited in $\mathcal{L}$ hash tables $\mathbf{T} = \{T_i\}_{=1}^{\mathcal{L}}$, each hash table T contains totally three kinds of data, 1) a G value set $\mathbf{G}$ of the original data, 2) a PQ code set $\mathbf{Q}$ of the original data and a sorted ID list $\mathbf{ID}'$ of the original data points. There are several other kinds of data, including the original data points and the compound hash keys which we are certain to generate, however all of them will only appear in the intermediate process of the index building, we do not finally maintain them in the index structure.

Fig. 5 depicts the construction procedure of the index structure. Overall, it consists of three phases, the ordering phase, the sorting-codes phase and the tree-building phase.

In the order phase, we firstly generate the compound hash key set $\mathbf{K}$ for all data points in $\mathbf{D}$ under a compound hash function $\mathcal{G}$. Then we convert them into G values and get the corresponding G value set $\mathbf{G}$. Next, we sort all the G values in ascending order according to the linear order defined in Equation 8 and get a sorted G value set $\mathbf{G}'$ together with the sorted ID list $\mathbf{ID}'$ of the original data points.

After we get the sorted G values, we come into the codes-sorting phase, in which we first generate the data codes $\mathbf{Q}$ for all the original data points under a pre-defined product quantizer. Then we rearrange them according to the order in $\mathbf{ID}'$ and get a sorted code set $\mathbf{Q}'$. $\mathbf{Q}'$ will finally be stored on the external memory. We specify s measured in machine words to denote the size of one PQ code, then the capacity of a B machine words disk page for PQ codes becomes:

$$V = \lfloor B/s \rfloor \tag{9}$$

The final phase is the tree-building. Different from SK-LSH, we maintain G values instead of compound hash keys in the B$^+$-trees. Note that, we do not maintain all the G values in $\mathbf{G}$, we extract a representative G value set $\mathbf{G}^R$ from it. Concretely, we represent each data page (full of PQ codes now) by extracting a pair of G values $\langle L, U \rangle$ which belong to the first and the last data code respectively on a certain disk page. Then a B$^+$-tree is built over all the representative G values pairs $\mathbf{G}^R$ as shown in Fig. 5. Note that the total number of data pages storing PQ codes is $\lceil n/V \rceil$, then the total number of the representative G values becomes:

$$R = 2\lceil n/V \rceil \approx 2ns/B \tag{10}$$

This means a compression ratio of nearly $B/(2s)$ of G value storage. Besides, $s \ll B$ usually holds, for example, 64-bit PQ codes in 4KB disk pages means $s = 2$ and $B = 1024$. Therefore, it is a significant space saving for the index structure while not degrade the routing effect of the B$^+$-tree in ANN search.

In conclusion, during the entire process of index constructing, we use one set of product quantizer to generate PQ codes and $\mathcal{L}$ set of compound hash functions for LSH encoding. Therefore, the index structure of SC-LSH contains $\mathcal{L}$ B$^+$-trees, $\mathcal{L}$ set of ID permutations and $\mathcal{L}$ set of relatively sorted data codes.

4.3 ANN Search

During the ANN search in SC-LSH, we set the termination condition as the number of data pages N_P allowed to be accessed in order to make a better control of both the accuracy and the I/O efficiency. As a result, we need to carefully determine the order of page accessing to make sure that more data items with higher quality can be reached during the search process. To achieve that, we firstly define the distance between a query point and a data page as follows:

Definition 7 (**Distance between a query point and a data page**) Given a query point q and a data page P_i containing at most V data codes, let G_q be the G value of q and a pair of G values $\langle L_i, U_i \rangle$ to denote P_i. The distance between the query point q and P_i, denoted as $dist_{Page}(q, P_i)$, is calculated as follows:

$$dist_{Page}(q, P_i) = \begin{cases} dist_G(G_q, L_i) & \text{if } G_q \leq_G L_i \\ 0 & \text{if } L_i \leq_G G_q \leq_G U_i \\ dist_G(G_q, U_i) & \text{if } U_i \leq_G G_q \end{cases} \tag{11}$$

As shown in Definition 7, for G_q and $\langle L_i, U_i \rangle$, there are three cases which need to be considered. In the first case, G_q is smaller than all the G values of the data items in P_i. So the lower bound of distances between G_q and the G values of data items in P_i is $dist_G(G_q, L_i)$. In the second case where G_q falls into the range between L_i and U_i, the distance of G_q and page P_i is defined as 0. In the third case where G_q is greater than all G values of data items in P_i, the lower bound is $dist_G(G_q, U_i)$.

Based on the above definition, we discuss in the following how to find the closest page to q in a sorted code set. Let n be the cardinality of $\mathbf{D}$ and $\rho = \lceil n/V \rceil$ be the number of data code pages. In addition, we use $\mathbf{P} = \{P_1, P_2, \ldots, P_\rho\}$ to refer to the all the data code pages. Note that, a B$^+$-tree in the index structure is created to index all the representative G values $\Theta = \{L_1, U_1, L_2, U_2, \ldots, L_\rho, U_\rho\}$. It is very convenient to find two G values in Θ, θ_i and θ_{i+1} $(1 \leq i \leq 2\rho - 1)$, such that $\theta_i \leq G_q \leq \theta_{i+1}$ by searching in the B$^+$-tree. If $\lceil \frac{i}{2} \rceil = \lceil \frac{i+1}{2} \rceil$, indicating that θ_i and θ_{i+1} belong to the same data page $P_{\lceil \frac{i}{2} \rceil}$, the closest data page to q is $P_{\lceil \frac{i}{2} \rceil}$ according to Definition 7. Otherwise, θ_i and θ_{i+1} belong to different data pages and we compare the distances of $P_{\lceil \frac{i}{2} \rceil}$ and $P_{\lceil \frac{i+1}{2} \rceil}$ to q respectively to figure out the closest one.

In SC-LSH, we employ $\mathcal{L}$ index files in order to reduce the loss of false negatives. The key operation for ANN search is to find the next data page to be accessed among $\mathcal{L}$ B$^+$-trees in the index files, which can be done by bi-directional expansion at data pages of all B$^+$-trees. Algorithm 1 summarizes the whole search process in multiple hash tables of SC-LSH.

Firstly, we set two pointers P_{iL} and P_{iR} on each hash table T_i to help guide the page accessing, as shown in line 2-5 in Algorithm 1 where $1 \leq i \leq \mathcal{L}$. We use P_{iL} to denote the closest data page to q in T_i, and P_{iR} to the data page immediately succeeding P_{iL}. Figure 6 shows an example where 3 index files are built. Taking T_1 as an example, P_{12} has the smallest distance to G_q^1. Hence, P_{1L} is set to be P_{12} and P_{13} as P_{1R}. Similarly, we obtain P_{2L}, P_{2R}, P_{3L} and P_{3R}.

ALGORITHM 1: *k*-ANN search on multi-trees

```
Require: hash tables $\mathbf{T} = \{T_i | 1 \le i \le \mathcal{L}\}$, query point $q$, number of
    returned nearest neighbors $k$, termination condition $N_P$;
Ensure: $k$ nearest neighbors $k$NN[];
 1: Initialize $o$=NULL, page counter $cnt$=0;
 2: for $i = 1$ to $\mathcal{L}$ do
 3:     Compute the G values $G_q^i$ of $q$;
 4:     Find out the initial pages $P_{iL}$ and $P_{iR}$, push them into $\Phi$;
 5: end for
 6: repeat
 7:     $P = extract(\Phi)$;
 8:     Verify data codes on page $P$ to update $k$NN[], make sure $k$NN[]
        keeps the data items with the smallest $k$ AQD to $q$ up to now;
 9:     $cnt$++;
10:     $P' = shift(P)$;
11:     $\Phi = \Phi \cup P'$;
12: until $cnt \ge N_p$
13: return $k$NN[];
```

It is easily derived that the page with the smallest distance to its corresponding G value of the query point must be in the set $\Phi = \{P_{1L}, P_{1R}, P_{2L}, P_{2R}, P_{3L}, P_{3R}\}$.

Line 6-12 in Algorithm 1 describes the loop where we constantly determine the next data page to load and verify the candidates among $\mathcal{L}$ hash tables. To facilitate the discussion, we define two operations on Φ, *extract* and *shift*. $extract(\Phi)$ returns the closest data page in Φ, say P^*, and P^* is meanwhile removed from Φ. $shift(P)$, where $P \in \Phi$, means moving the pointer to P away from q by one page to P'. For instance, the result of $shift(P_{1L})$ is P_{11} and that of $shift(P_{2R})$ is P_{24}. Usually, we will use the new page $shift(P)$ to replace P in Φ as shown in line 10-11. By repeating *extract* and *shift* operations, close data pages to the query point will be found from all B$^+$-trees in sequence. Before we do candidate refining, we need to prepare a lookup table composed of the squared distances between each sub vector of the query point and each sub codewords. After we load the candidate PQ codes into the memory, we perform AQD computations between the codes and the query according to Equation 4 based on the lookup table to determining nearest neighbors (line 8).

Finally, if the number of data pages verified exceeds a threshold, N_P, specified before the search, SC-LSH terminates the search process (line 12 in Algorithm 1). Additionally, we recommend to maintain a bitmap in the algorithm to avoid unnecessary replicated verification.

4.4 Complexity analysis

As mentioned in Section 4.2, the space consumption of SC-LSH mainly consists of three parts, 1) parameters of $\mathcal{L}$ set of compound hash functions, totally $\mathcal{L}md$, 2) $\mathcal{L}$ set of sorted ID list, totally $\mathcal{L}n$ and 3) $\mathcal{L}$ hash tables where each contains a B$^+$-tree maintaining a set of representative G values and one set of data codes. According to Equation 10, there are

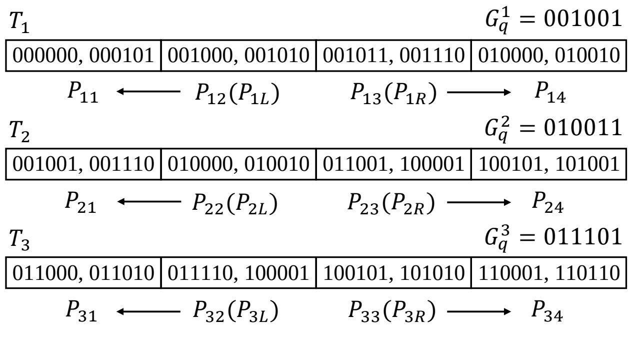


Fig. 6: Bi-directional expansion

totally about $R = 2ns/B$ representative G values to maintain in a B$^+$-tree. Therefore, the total space consumption of SC-LSH will be $O(\mathcal{L}(md + n + ns(2m/B + 1)))$. Usually $2m \ll B$ holds in practical cases, then the space consumption will become $O(\mathcal{L}(md + n(s + 1)))$. Note that this is a remarkable progress compared with SK-LSH since we make the space consumption get rid of the limitation from d.

The time consumption of SC-LSH consists of two parts, 1) disk processing among $\mathcal{L}$ hash tables to load N_P pages and 2) verifying the loaded candidate codes. The first part consists of the searching in $\mathcal{L}$ hash tables and loading candidate codes in N_P pages. The former one is related to the height E of the B$^+$-trees. Denote the order (or branch factor) of a B$^+$-tree as:

$$\beta = \lfloor B/m \rfloor \tag{12}$$

Then

$$E = \log_\beta R. \tag{13}$$

Since the time cost of processing a page is $O(B)$. The time cost for the first part is totally $O(\mathcal{L}\beta \log_\beta \frac{2ns}{B} + BN_P)$ combined with Equation 10. As for the candidates verification, the time cost is $O(BN_P\mathcal{M}/s)$ since each PQ code consumes an $O(\mathcal{M})$ AQD computation. Therefore, compared with SK-LSH, under the same amount of N_P pages, SC-LSH will reduce the time cost of disk page processing since $s \ll d$. However, we cannot draw a deterministic conclusion telling which is better in the candidate verification. Though an AQD computation is less expensive than an Euclidean distance computation, SC-LSH takes more verification tasks under the same disk page access.

Finally, the I/O cost in SC-LSH is quite explicit. It contains the nearest data page locating among $\mathcal{L}$ hash tables and N_P pages accessing, which is totally $\mathcal{L}E + N_P$.

5 Experiments

In this section, we empirically evaluate the performance of SC-LSH for ANN search. We choose three public real-world multimedia datasets to conduct the experiments, which are described as follows.

Audio[5]. Containing 54,387 192-D audio feature vectors extracted from the DARPA TIMIT audio speech database by the Marsyas library. We set *B* to be 1024 machine words.

Sift[6]. Containing 1,000,000 local SIFT [28] descriptors, the dimensionality of each vector is 128. *B* is set to be 1024 words.

Gist[7]. Consists of 1,000,000 960-D GIST [33] descriptors. *B* is set to be 4096 words.

For each dataset, we randomly select 200 points to form the query set and the rest as the base set.

5.1 Performance measures

We mainly employ the following measures to evaluate the accuracy and efficiency of LSH-based methods.

- **ratio**. *ratio* is used to evaluate the accuracy of the returned neighbors. Given a query q, let $o_1^*, o_2^*, \ldots, o_k^*$ be the kNNs w.r.t q, an ANN approach returns k points $o_1, o_2, \ldots, o_k$. Both results are ranked by the increasing order of their distance to q. Therefore, the approximate ratio for ANN w.r.t q is computed as

$$ratio(q) = \frac{1}{k} \sum_{i=1}^{k} \frac{\|o_i, q\|}{\|o_i^*, q\|} \tag{14}$$

  We use the mean of $ratio(q)$ over the query set.
- **I/O cost**. The I/O cost is defined as the number of pages accessed during the ANN search. For most external ANN search methods, I/O cost is treated carefully since it is somewhat one of the major factor that determine the speed of the ANN search.
- **Average Response Time (ART)**. Response time refers to the total running time of a method to accomplish the ANN search for a query point. In our experiments, we measure the average response time of all tested queries.
- **Space Consumption**. In our experiment, the space consumption of a LSH scheme is considered as the size of all the data that could be used during the ANN search.

5.2 Effect of better linear order

In this section we validate the effect of optimized linear order in SC-LSH. For that, we instantiate a halfway version of SC-LSH, SK-Gray, where we only introduce the Gray curve into the keys sorting yet store the original data points to do candidate refining. We compare it with SK-LSH to see the different effects on ANN search under different kinds of linear orders. Fig. 7(a) and Fig. 7(b) show the comparison results on the Sift dataset.

Fig. 7(a) compares the *ratio* versus different width of LSH functions. Note that we tune W with the ratio of 10 times. It is obviously that SK-LSH is very *sensitive* to W. Its accuracy becomes better when the width becomes larger and reaches the optima with a relative large width (in this case, W=1000 is the optimal width), after that, it becomes worse again. Relatively, SK-Gray seems not affected by the change of W, its accuracy stabilizes at a relatively good level (nearly equal to the optima of SK-LSH). The difference in the sensibility w.r.t W is derived from the different linear orders conducted in these two methods. As analyzed in Section 3.1.2, the row-wise order is more prone to miss near neighbors during the local traversing due to its dimension-first-traverse property. Enlarging the internal width of LSH functions helps to catch back those missed near neighbors. As for the Gray curve, it will always traverse nearer neighbors first before further areas no matter what the width is, therefore it does need to seek extra help.

A consequence of enlarging W in SK-LSH is that it has to build more LSH functions. Because larger W will enlarge the collision probability of two distant points according to Equation 2, which will bring in more false positives into the candidate set. To protect the accuracy, more LSH functions have to built to adequately filter out these false positives, which finally increases the index and computation overhead. Experiment results in Fig. 7(b) have corroborated this. As shown, within a certain range, adding LSH functions can help both the methods to improve the ANN search accuracy. The promotion degrades with the growth of m until a certain value after which adding more LSH functions no longer improve the accuracy. That is the optimal m value for each of the method. We see in the figure, SK-LSH has a larger optimal m=20 than that of SK-Gray which is 10.

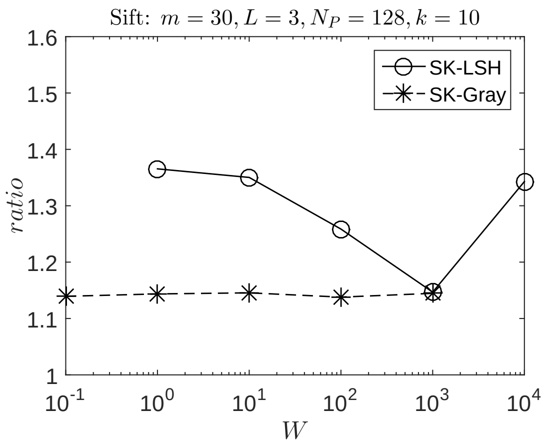
Note that there is a inflexion in the experiment results in Fig. 7(a), when W is larger than 1,000, the *ratio* of SK-LSH becomes worse again. This is because increasing W will decrease the number of intervals divided along a LSH function. For SK-LSH, W that larger than 1,000 will barely divide no more than 2 intervals which makes all data points mix together according to their compound hash keys.

In conclusion, employing better linear order does help to enhance the local distribution of NN candidates and thus boost the efficiency as well as the algorithm robustness.

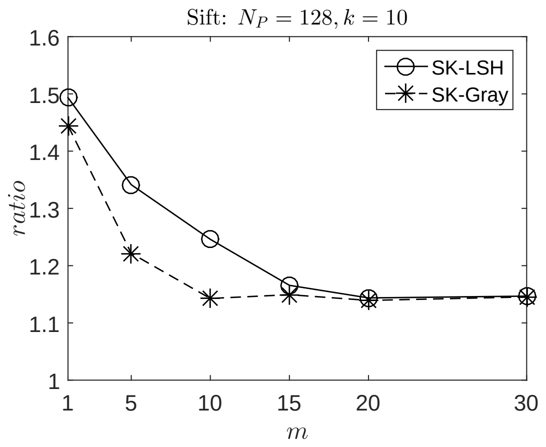
5.3 Performance Comparisons

In this section, we make thorough comparisons between SC-LSH and four other external methods which are the state-of-the-art LSH-based methods, including C2LSH [13], SK-LSH [27], SRS [38] and QALSH [19].

[5] http://www.cs.princeton.edu/cass/audio.tar.gz
[6] http://corpus-texmex.irisa.fr/
[7] http://corpus-texmex.irisa.fr/



(a) Comparison of different orders versus W



(b) Comparison of different orders versus m

Fig. 7: ANN search comparison on different linear orders

Table 1: Parameter settings

Datasets	C2LSH	QALSH	SRS	SK-LSH	SC-LSH
Audio	c=3, Ct=1	c=3	c=4	m=10, W=3, $\mathcal{L}$=3	m=10, W=1, $\mathcal{L}$=3
Sift	c=3, Ct=1	c=3	c=4	m=10, W=1000, $\mathcal{L}$=3	m=10, W=1, $\mathcal{L}$=3
Gist	c=3, Ct=1	c=3	c=4	m=10, W=2.5, $\mathcal{L}$=3	m=10, W=1, $\mathcal{L}$=3

5.3.1 Parameter Settings

To make a fair comparison, we set the parameters of each methods to exhibit their best tradeoff on both the accuracy and the I/O efficiency. The detailed parameter settings are listed in Table 1.

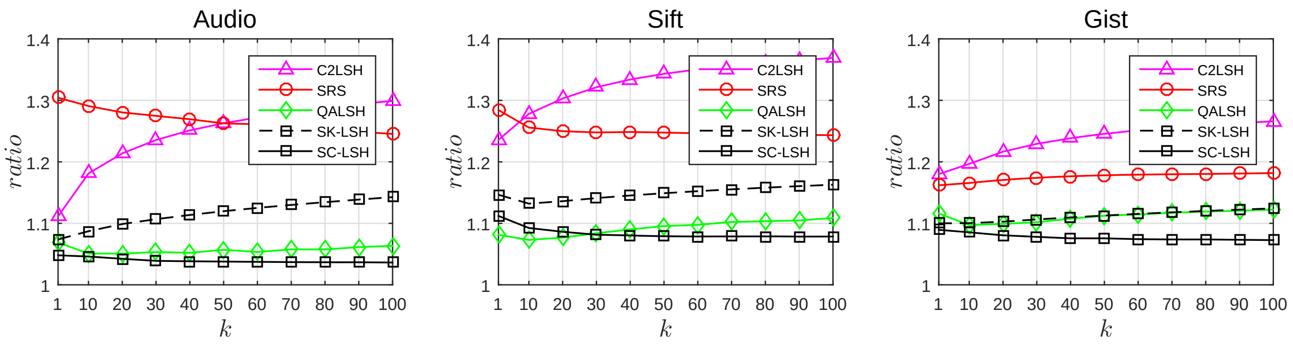
To some extent, C2LSH and QALSH are similar to each other since they are both based on the dynamic collision counting mechanism. We specify them to achieve different goals. We turn on the early termination condition (Ct) in C2LSH to achieve better efficiency, while we let QALSH pursuit better accuracy. As for SRS, we deploy the SRS−12 variant and set c=4 on all datasets as suggested in [38]. For SK-LSH, it needs to tune the parameters to express the best performance, especially the width W of LSH functions. Finally, W are fixed at 3, 1000, 2.5 for Audio, Sift and Gist, respectively. SC-LSH is not sensitive with W, in this paper, we fix W=1 for all datasets. Actually, in our practical experimental observations, any small value that can divide more than 3~4 internals along the LSH functions is reasonable for SC-LSH. Besides, we set m=10 and $\mathcal{L}$=3 for both SK-LSH and SC-LSH on all the datasets for the sake of fairness.

We vary the number k of the returned nearest neighbor among $\{1, 10, 20, \ldots, 100\}$ to observe the performance trend of all methods for ANN search and list all the experimental results in Fig. 8 and Table 2 and Table 3.

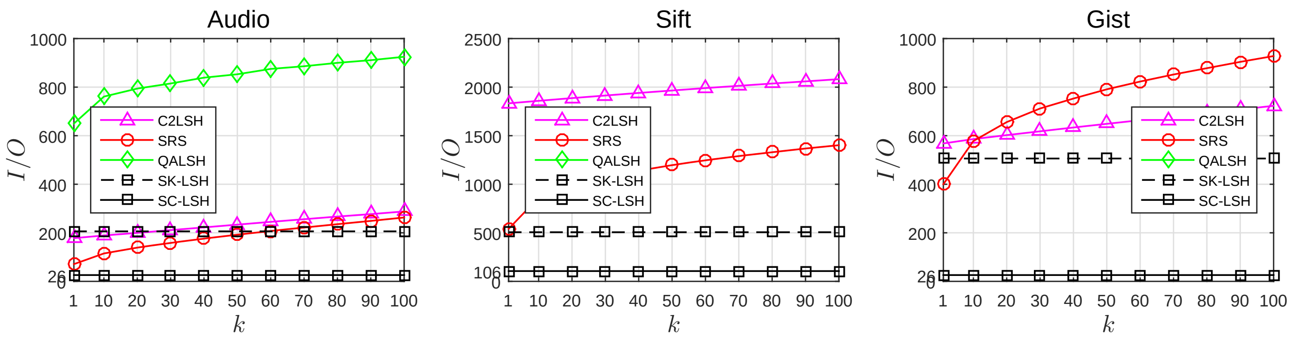
5.3.2 Accuracy and I/O efficiency

Accuracy. According to Fig. 8(a), SC-LSH achieves the best accuracy on all datasets, following is QALSH and SK-LSH and then SRS and C2LSH. The result reflects that the similarity-preserving ability of PQ codes is trustworthy in candidate refining and also demonstrates our opinion that discriminating ability is the key of nearest neighbor discriminating and we are not necessarily relied on the original distance to do candidate refining as discussed in Section 3.2. Besides, a characteristic of the ANN search accuracy of SC-LSH is that the accuracy is slightly worse when k is small and becomes better with the growth of k. This is caused by the quantization distortion in PQ codes, because nearer neighbors (i.e., when k is small) are more likely to be affected by the quantization distortions compared with further ones.

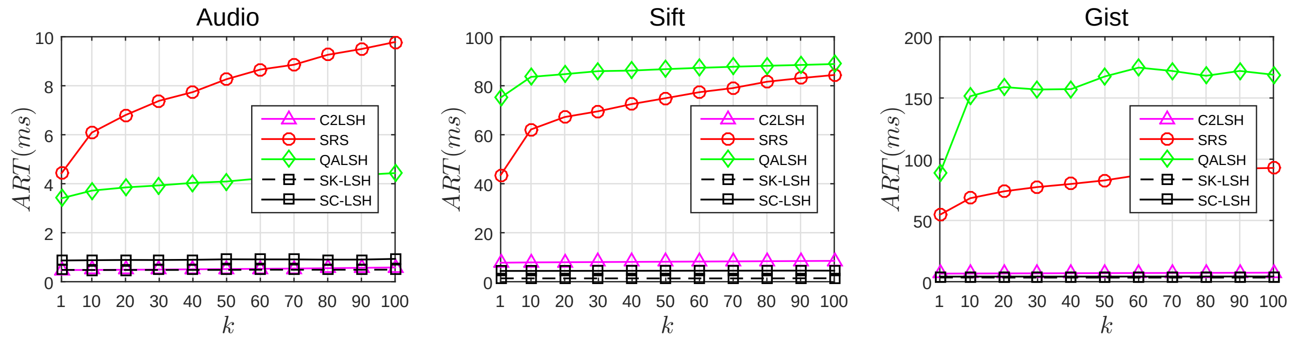
I/O efficiency. Considering the I/O cost in Fig. 8(b), SC-LSH also achieves the best performance, following is SK-LSH and then SRS and C2LSH. Quantitatively, SC-LSH saves 87%, 78% and 95% I/O operations than SK-LSH respectively on the three datasets. This is due to the excellent space efficiency of PQ codes, which enables SC-LSH to access sufficiently high quality candidates in much fewer disk page accesses. In contrast, since the candidate capacity of the disk page in SK-LSH is limited by the data dimensionality and usually the disk page size is constant



(a) Comparison of *ratio*



(b) Comparison of I/O cost



(c) Comparison of *ART*

Fig. 8: Experimental comparisons

Table 2: Comparison of I/O cost between QALSH and SC-LSH

k	Audio			Sift			Gist		
	QALSH	SC-LSH	Q/S	QALSH	SC-LSH	Q/S	QALSH	SC-LSH	Q/S
1	650	26	25	11901	106	112	3659	26	140
10	761	26	29	13298	106	125	4052	26	155
20	794	26	30	13512	106	127	4123	26	158
30	815	26	31	13646	106	128	4166	26	160
40	839	26	32	13753	106	129	4192	26	161
50	853	26	32	13836	106	130	4216	26	162
60	875	26	33	13926	106	131	4244	26	163
70	886	26	34	13986	106	131	4268	26	164
80	900	26	34	14056	106	132	4290	26	165
90	911	26	35	14128	106	133	4313	26	165
100	925	26	35	14174	106	133	4332	26	166

in a certain machine, it has to consume more disk pages in high dimensional datasets. Note that SK-LSH consumes the same amount of I/O operations (i.e., 506) on both Sift and Gist according to Fig. 8(b), it seems that SK-LSH is not affected by the data dimensionality. Actually, this is because we set different page size for this two datasets, 4KB for Sift and 16KB for Gist, respectively. Besides, we do not draw all the I/O cost of QALSH in Fig. 8(b), because it consumes much more I/O operations than the other methods which can be reflected by the results on the Audio dataset in Fig. 8(b) in a certain extent. Actually, it consumes more I/O operations on other two larger-scaled datasets. We list the numerical results in an extra table (Table 2) to better show the difference. As can be seen, on the small-scaled Audio dataset, QALSH consumes 25-35 times more page accesses than SC-LSH. On the two larger datasets, Sift and Gist, the difference grows to 112-133 and 140-166 times, respectively.

Accuracy + I/O efficiency. Considering both the accuracy and I/O efficiency, which are the most two concerned performances in this paper, SC-LSH still performs the best. QALSH achieves the most proximate accuracy with SC-LSH, sometimes even better (e.g., for some smaller k on Sift), however, it costs the most I/O operations. SK-LSH achieves a relatively good tradeoff between accuracy and I/O efficiency, however, due to the limitation of data dimensionality, it falls back SC-LSH in all these two performances. The SRS method and the early stop version of C2LSH can neither reconcile the contradiction between accuracy and I/O efficiency.

5.3.3 Time and space consumption

As for the *ART* in Fig. 8(c), it shows that SC-LSH is not the fastest, SK-LSH consumes a bit less time than SC-LSH. The reason, we have mentioned in the discussion of the time complexity in Section 4.4. Although an AQD computation is less expensive than an Euclidean distance computation, SC-LSH takes more verifications under same amount of disk page accesses. For example, on Audio, SK-LSH can get about 1,000 points in 200 pages, while SC-LSH can access nearly 10,000 data codes even in 20 pages. Besides, the significant I/O consumption leads QALSH to the most ANN search time on Sift and Gist as shown in Fig. 8(c).

Table 3 lists the relative space consumption of all the methods. As can be seen, the maintenance of original datasets has become the dominant storage overhead of the methods. For example, SK-LSH maintains the most copies of original datasets ($\mathcal{L}$=3 in our experiments), as a result, it has the most space consumption among all compared methods. Followed are C2LSH, QALSH and SRS. SRS keeps its leading position in space saving among all the methods maintaining original datasets due to its tiny index structure size. SC-LSH no longer needs original data to do candidate refining, therefore it has the least space consumption.

Table 3: Comparison of Space Consumption

Methods	**Audio** (39.3MB)	**Sift** (492.01MB)	**Gist** (3.58GB)
C2LSH	135MB	1.43GB	4.55GB
QALSH	49MB	635MB	3.96GB
SRS	42MB	537MB	3.71GB
SK-LSH	121MB	1.47GB	10.80GB
SC-LSH	4MB	103MB	103MB

6 Related Work

There is a large body of literature on the topic of NN search. Early studies proposed a plenty of work that seek for *exact* nearest neighbors, however, all of them suffer from the “curse of dimensionality” [4,46]: their running time for NN search degenerate to that of linear scan. Recent developed methods have made some progress in overcoming the dimension curse, such as [46,10,22,37,9,20], etc. However, to guarantee the strict 100% accuracy, most of them still have to consume large amount of I/O operations and distance computations which makes the efficiency far from satisfactory.

Approximate nearest neighbor (ANN) search seeks for high-quality approximate solutions which can fulfill most the query requirements. Without the strict restriction of search accuracy, plenty of effective techniques are developed to promote the search efficiency. Typically, there are three kinds of techniques most developed, which are the hashing techniques, the quantization techniques and the graph based techniques.

6.1 Hashing techniques

Hashing techniques and in particular locality-sensitive hashing (LSH) have recently gained its popularity in the communities of database and computer vision due to the high efficiency. LSH employs distance-preserving hashing functions to project nearby points into same bucket with high probability which enables fast and accurate irrelevant points filtration with barely no pre-processing. Associated with the construction of compound hash functions and/or multiple hash tables, researchers can effectively reduce the false positives (FP) and false negatives (FN).

The basic LSH methods were first developed in [21,15, 5] by Indyk, Gionis and Datar et al. They consume too much space consumption and are only designed for main memory adoption. More works are developed to further boost LSH-based methods. Some of them devote to reduce the space consumption, including Entropy-based LSH [34] and Multi-Probe LSH [29,25]. Tao et al. propose LSB technique [40] which is the first LSH method designed for disk-resident data. It generates z-orders with the use of hash keys of compound LSH functions and employs B-trees, called LSB-forest, to manage the index files, which produces a good tradeoff between quality and efficiency. C2LSH [13] uses *dynamic collision counting* to estimate the actual distances between two points and only a few number of candidates are satisfactory to obtain high-quality results. Also based on the dynamic collision counting mechanism, QALSH introduces a concept of *query-aware* bucket partition which uses the query as the “anchor” for bucket partition after the query arrives so that to remove the random shift in the traditional *query-oblivious* projection. SRS aimes at reducing the index size, it uses only 6 random projections to convert high-dimensional data into low dimensions so that they can be indexed by a single tiny index. Most the above methods unavoidably need to consume large number of random I/Os for candidates collecting, SK-LSH proposes a sorting-keys strategy to address this. It introduces space linear order into compound hash keys to enhance the local distribution of candidates which improves the disk access manner as well as loading more high-quality candidates during a single page access. However, this measure unfortunately endures poor scalability with high dimensions. Since the page size B is usually constant, as the dimensionality grows, less candidates can be held in a single disk page, which obliges it ultimately resort for more disk page access to guarantee the search accuracy.

In fact, the LSB technique is the first one who exploits linear order to estimate the similarity between data points. But it mainly emphasizes on how to boost the ANN accuracy and does not provide further measurements on how to optimize I/O operations as what SK-LSH does. Nevertheless, thanks to their instructive work. It is during the exploring of the difference between row-wise curve and z-ordering where we come to realize the difference effects on the local arrangement of candidates between DFT orders and NFT ones.

6.2 Quantization techniques

Quantization techniques, especially product quantization [23, 32] are increasingly popular in the community of computer vision. They encode all points with a product quantizer and compute *asymmetric quantizer distances (AQD)* between codes of query and data points during the search process. Finally, the data point with the smallest AQD value is returned as the ANN of the query. The basic PQ method was proposed by H. Jegou et al. [23] and has been improved by several works [2,32,44,26] since then. In particular, M. Norouzi et al. [32] proposed CK-Means, which improves the accuracy of returned results by optimally rotating the original space and re-ordering the dimensions.

Although PQ codes are initially designed for internal memory, we explore its potential in external memory in this paper to sharpen our index for ANN search. Thanks to PQ-based technique, SC-LSH is now insensitive to the dimensionality of dataset. It can verify much more candidate points than [27] within the same or even less number of I/O accesses. As a result, SC-LSH can preserve similar, or even dramatically higher, accuracy under fewer I/O operations. We hope that our exploration can provide valuable references for future research.

6.3 Graph based techniques

There is another series of work based on neighborhood graph. Neighborhood graph (or nearest neighbor graph, NNG) is a directed graph that connects each data point to its nearest neighbors. Searching in a neighborhood graph is often proceeded by a local neighborhood graph search procedure that incorporates various heuristic rules such as best-first strategy or priority queue. A fundamental limitation of the local search is that the search process may enter into local optima. Mainly two kinds of efforts are devoted to alleviate this problem. The first one focuses on optimizing the search strategy, including improving the initial candidate neighbors [17,24] or better guiding the iterated neighborhood graph expansion [43,45,31,11]. Another efforts reside in sharpening the neighborhood graph construction. Building a k-nearest neighbor graph (k-NNG) is the most direct way to approximate the Delaunay graph which can maintain efficient exploration costs by limiting the degree of each vertex in the graph [6,35]. Approximate k-nearest neighbor graph [17,36,14,48] provides an alternative approach to approximate the k-NNG which can offer a significant speed-up for the offline graph building. Besides, there are some other methods seek for adaptively neighbor connecting method in order to exploit some of the intrinsic structure of the dataset [30,18].

Overall, graph-based techniques have made significant progress in boosting the ANN search accuracy but have also left over two main issues. Firstly, computational cost of constructing neighborhood graph is usually very large which should be treated cautiously. Besides, routing on the neighborhood graph relied on original distances, the access of data points are all consumed in random I/Os.

Some of the latest work also concern about the I/O performance boosting of external ANN search methods. A representative one is made by Tang et al. who manage to reduce the candidate refining time as well as the I/O cost by exploiting caching techniques of compact approximate representations of data points in main memory, which is orthogonal to our work [39].

7 Conclusion

In this paper, we propose a novel LSH index structure, SC-LSH, to further boost the ANN search accuracy together with the I/O efficiency. We exploit the Gray curve with neighbor-first-traverse property as the order of compound hash keys sorting to enhance the local distribution of NN candidates and thus boost the ANN search efficiency as well as the algorithm robustness. Storing compact binary codes (specifically PQ codes) instead of original data points to do candidate refining can significantly reduce the I/O cost, and extensive experiments conducted on real-life datasets demonstrate the ability of PQ codes for NN discriminating and also the superiority of SC-LSH over the other compared LSH approaches.

Acknowledgements This work is supported by National Natural Science Foundation of China (Nos. 61472298, 61672408, 61702403, U1135002), National High Technology Research and Development Program (863 Program, No.2015AA016007), China 111 Project (No. B16037), Natural Science Basic Research Plan in Shaanxi Province of China (Program No. 2015JQ6227), SRF for ROCS, SEM and the Fundamental Research Funds for the Central Universities (JB170308).

Acknowledgements This work is supported by National Natural Science Foundation of China (Nos. 61472298, 61672408, 61702403, U1135002), National High Technology Research and Development Program (863 Program, No.2015AA016007), China 111 Project (No. B16037), Natural Science Basic Research Plan in Shaanxi Province of China (Program No. 2015JQ6227), SRF for ROCS, SEM and the Fundamental Research Funds for the Central Universities (JB170308).

References

1. André, F., Kermarrec, A.M., Le Scouarnec, N.: Cache locality is not enough: high-performance nearest neighbor search with product quantization fast scan. Proceedings of the VLDB Endowment **9**(4), 288–299 (2015)
2. Babenko, A., Lempitsky, V.: The inverted multi-index. In: Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on, pp. 3069–3076. IEEE (2012)
3. Böhm, C.: A cost model for query processing in high dimensional data spaces. ACM Trans. Database Syst. **25**(2), 129–178 (2000)
4. Böhm, C., Berchtold, S., Keim, D.A.: Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. ACM Comput. Surv. **33**(3), 322–373 (2001)
5. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: SoCG, pp. 253–262 (2004)
6. Dong, W., Moses, C., Li, K.: Efficient k-nearest neighbor graph construction for generic similarity measures. In: International Conference on World Wide Web, pp. 577–586 (2011)
7. Faloutsos, C.: Multiattribute hashing using gray codes. In: ACM SIGMOD International Conference on Management of Data, Washington, D.c., May, pp. 227–238 (1986)
8. Faloutsos, C., Roseman, S.: Fractals for secondary key retrieval. In: Eighth ACM Sigact-Sigmod-Sigart Symposium on Principles of Database Systems, pp. 247–252 (1989)
9. Feng, X., Cui, J., Liu, Y., Li, H.: Effective optimizations of cluster-based nearest neighbor search in high-dimensional space. Multimedia Syst. **23**(1), 139–153 (2017)
10. Ferhatosmanoglu, H., Tuncel, E., Agrawal, D.: Vector approximation based indexing for non-uniform high dimensional data sets. In: CIKM, pp. 202–209 (2000)
11. Fu, C., Cai, D.: EFANNA : An extremely fast approximate nearest neighbor search algorithm based on knn graph. CoRR **abs/1609.07228** (2016)
12. Gaede, V., Günther, O.: Multidimensional access methods. ACM Computing Surveys (CSUR) **30**(2), 170–231 (1998)
13. Gan, J., Feng, J., Fang, Q., Ng, W.: Locality sensitive hashing scheme based on dyanmic collision counting. In: SIGMOD, pp. 541–552 (2012)
14. Gan, R.: Scalable k-nn graph construction for visual descriptors. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 1106–1113 (2012)
15. Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. In: VLDB, pp. 518–529 (1999)
16. Gong, Y., Lazebnik, S.: Iterative quantization: A procrustean approach to learning binary codes. In: Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on, pp. 817–824 (2011)
17. Hajebi, K., Abbasi-Yadkori, Y., Shahbazi, H., Zhang, H.: Fast approximate nearest-neighbor search with k-nearest neighbor graph. In: IJCAI 2011, Proceedings of the International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July, pp. 1312–1317 (2009)
18. Harwood, B., Drummond, T.: FANNG: fast approximate nearest neighbour graphs. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016, pp. 5713–5722 (2016)
19. Huang, Q., Feng, J., Zhang, Y., Fang, Q., Ng, W.: Query-aware locality-sensitive hashing for approximate nearest neighbor search. Proceedings of the Vldb Endowment **9**(1), 1–12 (2015)
20. Hwang, Y., Han, B., Ahn, H.: A fast nearest neighbor search algorithm by nonlinear embedding. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012, pp. 3053–3060 (2012)
21. Indyk, P., Motwani, R.: Approximate nearest neighbors: Towards removing the curse of dimensionality. In: Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998, pp. 604–613 (1998)
22. Jagadish, H.V., Ooi, B.C., Tan, K.L., Yu, C., Zhang, R.: idistance: An adaptive b+-tree based indexing method for nearest neighbor search. ACM Trans. Database Syst. **30**(2), 364–397 (2005)
23. Jegou, H., Douze, M., Schmid., C.: Product quantization for nearest neighbor search. IEEE TPAMI **33(1)**, 117–128 (2011)
24. Jin, Z., Zhang, D., Hu, Y., Lin, S., Cai, D., He, X.: Fast and accurate hashing via iterative nearest neighbors expansion. IEEE Trans. Cybernetics **44**(11), 2167–2177 (2014)
25. Joly, A., Buisson, O.: A posteriori multi-probe locality sensitive hashing. In: ACM Multimedia, pp. 209–218 (2008)
26. Kalantidis, Y., Avrithis, Y.S.: Locally optimized product quantization for approximate nearest neighbor search. In: 2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014, pp. 2329–2336 (2014)
27. Liu, Y., Cui, J., Huang, Z., Li, H., Shen, H.T.: SK-LSH: an efficient index structure for approximate nearest neighbor search. PVLDB **7**(9), 745–756 (2014)
28. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision **60**(2), 91–110 (2004)
29. Lv, Q., Josephson, W., Wang, Z., Charikar, M., Li, K.: Multi-probe lsh: efficient indexing for high-dimensional similarity search. In: VLDB, pp. 950–961 (2007)

30. Malkov, Y., Ponomarenko, A., Logvinov, A., Krylov, V.: Approximate nearest neighbor algorithm based on navigable small world graphs. Inf. Syst. **45**, 61–68 (2014)
31. Malkov, Y.A., Yashunin, D.A.: Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. CoRR **abs/1603.09320** (2016)
32. Norouzi, M., Fleet, D.J.: Cartesian k-means. In: 2013 IEEE Conference on Computer Vision and Pattern Recognition, Portland, OR, USA, June 23-28, 2013, pp. 3017–3024 (2013)
33. Oliva, A., Torralba, A.: Modeling the shape of the scene: A holistic representation of the spatial envelope. International Journal of Computer Vision **42**(3), 145–175 (2001)
34. Panigrahy, R.: Entropy based nearest neighbor search in high dimensions. In: SODA, pp. 1186 – 1195 (2006)
35. Paredes, R., Chvez, E., Figueroa, K., Navarro, G.: Practical Construction of k-Nearest Neighbor Graphs in Metric Spaces. Springer Berlin Heidelberg (2006)
36. Ponomarenko, A., Avrelin, N., Naidan, B., Boytsov, L.: Comparative analysis of data structures for approximate nearest neighbor search. In: International Conference on Emerging Network Intelligence, pp. 125–130 (2011)
37. Ramaswamy, S., Rose, K.: Adaptive cluster distance bounding for high-dimensional indexing. IEEE Trans. on Knowl. and Data Eng. **23**(6), 815–830 (2011)
38. Sun, Y., Wang, W., Qin, J., Zhang, Y., Lin, X.: SRS: solving c-approximate nearest neighbor queries in high dimensional euclidean space with a tiny index. PVLDB **8**(1), 1–12 (2014)
39. Tang, B., Man, L.Y., Hua, K.A.: Exploit every bit: Effective caching for high-dimensional nearest neighbor search. IEEE Transactions on Knowledge and Data Engineering **28**(5), 1175–1188 (2016)
40. Tao, Y., Yi, K., Sheng, C., Kalnis, P.: Quality and efficiency in high dimensional nearest neighbor search. In: SIGMOD, pp. 563–576 (2009)
41. Torralba, A., Fergus, R., Weiss, Y.: Small codes and large image databases for recognition. In: Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on, pp. 1–8 (2008)
42. Vitter, J.S.: Algorithms and data structures for external memory. Foundations and Trends® in Theoretical Computer Science **2**(4), 305–474 (2008)
43. Wang, J., Li, S.: Query-driven iterated neighborhood graph search for large scale indexing. In: ACM International Conference on Multimedia, pp. 179–188 (2012)
44. Wang, J., Wang, J., Song, J., Xu, X., Shen, H.T., Li, S.: Optimized cartesian k-means. IEEE Trans. Knowl. Data Eng. **27**(1), 180–192 (2015)
45. Wang, J., Wang, J., Zeng, G., Gan, R., Li, S., Guo, B.: Fast neighborhood graph search using cartesian concatenation. In: IEEE International Conference on Computer Vision, pp. 2128–2135 (2013)
46. Weber, R., Schek, H.J., Blott, S.: A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In: VLDB, vol. 98, pp. 194–205 (1998)
47. Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: In Proc. NIPS (2008)
48. Zhang, Y.M., Huang, K., Geng, G., Liu, C.L.: Fast knn graph construction with locality sensitive hashing. In: Proceedings, Part II, of the European Conference on Machine Learning and Knowledge Discovery in Databases - Volume 8189, pp. 660–674 (2013)