SK-LSH : An Efficient Index Structure for Approximate Nearest Neighbor Search

Yingfan Liu‡, Jiangtao Cui‡, Zi Huang§, Hui Li‡, Heng Tao Shen§

‡*School of Computer, Xidian University, China*
yfliu1989@gmail.com, {cuijt,hli}@xidian.edu.cn

§*School of Information Technology and Electrical Engineering, University of Queensland, Australia*
{huang, shenht}@itee.uq.edu.au

ABSTRACT

Approximate Nearest Neighbor (ANN) search in high dimensional space has become a fundamental paradigm in many applications. Recently, Locality Sensitive Hashing (LSH) and its variants are acknowledged as the most promising solutions to ANN search. However, state-of-the-art LSH approaches suffer from a drawback: accesses to candidate objects require a large number of *random* I/O operations. In order to guarantee the quality of returned results, sufficient objects should be verified, which would consume enormous I/O cost.

To address this issue, we propose a novel method, called SortingKeys-LSH (SK-LSH), which reduces the number of page accesses through locally arranging candidate objects. We firstly define a new measure to evaluate the distance between the compound hash keys of two points. A linear order relationship on the set of compound hash keys is then created, and the corresponding data points can be sorted accordingly. Hence, data points that are close to each other according to the distance measure can be stored locally in an index file. During the ANN search, only a limited number of disk pages among few index files are necessary to be accessed for sufficient candidate generation and verification, which not only significantly reduces the response time but also improves the accuracy of the returned results. Our exhaustive empirical study over several real-world data sets demonstrates the superior efficiency and accuracy of SK-LSH for the ANN search, compared with state-of-the-art methods, including LSB, C2LSH and CK-Means.

1. INTRODUCTION

Nearest Neighbor (NN) search is an important problem in many multimedia applications. The majority of multimedia data, such as images, audio and video clips can be represented as high-dimensional local/global feature vectors [3]. Thus, finding a multimedia object that is similar to a given query is converted to an NN search in the corresponding feature vector space, which aims at returning the closest vector (or point) to the query vector according to a particular distance measure (usually Euclidean distance). Due to the notorious "Curse of Dimensionality", the performance of most existing methods [1, 11, 21] for exact NN search degenerates as the dimensionality increases and is eventually outperformed by the brute-force approach, linear-scan [20]. To find an efficient solution to NN search, many researchers have recently focused on approximate nearest neighbor (ANN) search, which aims to return a point *close enough* to a query point instead of the closest one.

Locality Sensitive Hashing (LSH) has been shown to be the most promising solution to ANN search. It employs distance-preserving hash functions to project nearby points into the same bucket with high probability [5, 8]. Formally, a hashing function is defined as $h : \mathbb{R}^d \rightarrow \mathcal{Z}$, where $\mathcal{Z}$ denotes the domain of integers. In order to improve the hashing effect, state-of-the-art hashing algorithms [2, 4, 19] utilize m randomly-chosen LSH functions together to generate a compound hash key with m elements for each point to distinguish them from each other.

After a few years' development, several approaches based on LSH have been proposed, including the basic LSH [2, 8, 5], Multi-Probe LSH [10, 12], LSB [19], C2LSH [4], etc. They use different strategies to organize the compound hash keys of points to determine candidate points. The basic LSH [2] determines the candidate points in a straightforward way, which only considers the points sharing the same compound hash key with the query point over a compound LSH function. In compensation for the loss of candidate points because of its strict filtering policy, hundreds of or even more hash tables are constructed, which causes a huge space requirement. To reduce the number of hash tables, Multi-Probe LSH [12] was proposed, which could find more similar points from a single hash table by exploring the buckets near the one into which the query point falls.

LSB [19] takes a complicated but effective strategy to evaluate the similarity between points with their corresponding compound hash keys. Points in the original space are projected into a new space $\mathcal{Z}^m$ associated with m randomly selected LSH functions, then the corresponding point in $\mathcal{Z}^m$ is converted into a string of bits, called z-order. Thus, the points will be accessed according to the similarities of their z-orders to that of the query point. C2LSH [4] proposes an interesting method to collect candidate points, called dynamic collision counting. At first, a base containing m (usually hundreds of) LSH functions is built, where each LSH function corresponds to a hash table. C2LSH selects points frequently colliding with the query point among the

Proceedings of the VLDB Endowment, Vol. 7, No. 9
Copyright 2014 VLDB Endowment 2150-8097/14/05.

base as candidate points. However, both LSB and C2LSH collect their candidates among a large number of hash structures, and their candidates are distributed among different disk (or data) pages. Thus, a large number of I/Os are unavoidable in order to obtain sufficient candidates to guarantee the satisfactory accuracy of the returned results.

Intuitively, the number of I/O accesses can be reduced if the candidates to be accessed are distributed locally. In other words, similar objects should be stored in consecutive disk pages. For this purpose, we propose a novel distance measure to estimate the distance (i.e. dissimilarity) between the compound hash keys of two points. We also prove that the distance measure guarantees the similarity between two points, in the sense that the probability of a close pair having a small distance between their corresponding compound hash keys is larger than that for a far pair. To further improve the probability, $\mathcal{L}$ compound LSH functions are employed, each of which corresponds to an index file, such that the number of false negatives can be reduced significantly. $\mathcal{L}$ is as small as 3 in our experiments. Then, a linear order relationship on compound hash keys is created to sort all compound hash keys as well as their corresponding points, which makes it possible that points with close compound hash keys according to our distance measure are distributed locally. Furthermore, based on our new distance measure and the linear order relationship of compound hash keys, we propose a novel index structure, called SortingKeys-LSH (SK-LSH), which verifies candidates in the unit of disk page. As points with close compound hash keys are arranged together in the disk space, only a small number of disk page accesses are required to find enough candidate points and return precise neighbors. By reducing random I/Os considerably, SK-LSH accelerates the search process significantly. In addition, SK-LSH is simple yet *effective*.

Empirical results on several real-life data sets (ranging from low- to high-dimensional data) show that SK-LSH exhibits the best performance with respect to the *result quality*, *response time* and *space requirement* comparing with state-of-the-art LSH methods, including LSB and C2LSH. According to our experiments, SK-LSH speeds up ANN search by an order of magnitude compared with LSB and C2LSH. In addition, we also compare SK-LSH with Cartesian k-means (CK-Means) [14], the state-of-the-art method based on Product Quantization (PQ), which is very popular in the community of computer vision. As shown in Section 5.5, SK-LSH costs far less time than CK-Means when returning results with similar accuracy.

Our contributions are summarized as follows:

- We propose a new method to measure the distance between the compound hash keys of two points, which can be used to estimate the actual distance of two points. We also prove the effectiveness of our distance measure.

- We propose a linear order relationship on the set of compound hash keys to sort the compound hash keys and their corresponding points. We also prove that points with close compound hash keys according to our distance measure can be distributed locally based on the proposed linear order.

- We propose a novel index structure called SK-LSH, which is able to consume a very small number of page accesses for high-quality results.

- We demonstrate the superiority of SK-LSH over state-of-the-art approaches by extensive experiments conducted on real-life data sets.

The rest of this paper is organized as follows. We review some related work in Section 2. In Section 3 we introduce LSH functions and propose a novel distance measure between the compound hash keys of two points and discuss the linear order relationship. In Section 4 we show the details of SK-LSH. Section 5 presents the experimental evaluation. Finally, we conclude this paper in Section 6.

2. RELATED WORK

There are a large number of methods proposed for ANN search. Among all those methods, three categories of approaches attract the most attention, including methods based on Dimensionality Reduction (DR), LSH, and PQ.

DR based methods [7, 16, 17] first project high-dimensional points into a much lower-dimensional space and then build indices in the projected space. Since NN search has been solved well in low-dimensional spaces, DR based methods could gain its efficiency by using those well-built approaches for low-dimensional spaces [1, 11, 21].

LSH based methods are the most popular methods in the communities of database and computer vision due to their efficiency and error guarantee. The basic LSH method was first proposed by M. Datar [2], but it is too space-consuming. Several methods were proposed to reduce the space requirement, including Entropy-based LSH [15] and Multi-Probe LSH [10, 12]. LSB [19] is the first LSH method that is designed for disk-resident data, followed by C2LSH [4], which improves the efficiency and accuracy and reduces the space requirement. Note that there also exist many machine learning based hashing methods [6, 18]. However, they usually require an expensive learning process to learn hash functions which could be highly data-dependent.

PQ based methods [9, 14] are increasingly popular in the community of computer vision. They assume that the main memory is large enough to contain the whole data set and its index file. They encode each point into a short string of bits with a product quantizer and compute asymmetric quantizer distances (AQD) between codes of points and that of the query during the search process. Finally, the point with the smallest AQD value is returned as the ANN of the query. The basic PQ method was proposed by H. Jegou [9] and has been improved by a few researchers. Among them, M. Norouzi [14] proposed CK-Means, which uses Cartesian k-means to improve the accuracy of the returned results.

3. DISTANCE AND LINEAR ORDER OVER COMPOUND HASH KEYS

In this section, we first give a brief introduction to the ANN problem and (compound) LSH functions, followed by the distance measure and linear order over compound hash keys based on the compound LSH functions.

Given a data set $D \subset \mathbb{R}^d$ and a query point q, the target of NN problem is to find the point $o^* \in D$ satisfying that for any point $p \in D$, $\| o^*, q \| \leq \| p, q \|$, 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 $o \in D$ that satisfies $\| o, q \| \leq c \| o^*, q \|$. Here, c is the approximation ratio, which exceeds 1.

3.1 Locality Sensitive Hashing

To solve the c-ANN problem, Indyk and Motwani proposed the idea of LSH [8], which is formally defined as follows.

Definition 1. ***(Locality Sensitive Hashing)*** *Given a distance R, an approximate ratio c and two probability values $\mathcal{P}_1$ and $\mathcal{P}_2$, a hash function $h : \mathbb{R}^d \rightarrow \mathcal{Z}$ is called $(R, c, \mathcal{P}_1, \mathcal{P}_2)$**-sensitive** if it satisfies the following conditions simultaneously for any two points $p_1, p_2 \in D$:*

- *If $\| p_1, p_2 \| \leq R$, then $Pr[h(p_1) = h(p_2)] \geq \mathcal{P}_1$;*
- *If $\| p_1, p_2 \| \geq cR$, then $Pr[h(p_1) = h(p_2)] \leq \mathcal{P}_2$;*

*To make sense, both $c > 1$ and $\mathcal{P}_1 \geq \mathcal{P}_2$ hold. In addition, a **compound LSH function** is denoted as $G = (h_1, \ldots, h_m)$, where $h_1, \ldots, h_m$ are randomly selected LSH functions. Specifically, for $\forall p \in D$, $K = G(p) = (h_1(p), \ldots, h_m(p))$ is defined as the **compound hash key** of point p under G.*

According to Definition 1, LSH ensures that a close pair collides with each other with a high probability ($\mathcal{P}_1$) and a far pair with a low probability ($\mathcal{P}_2$). This property of LSH is also called distance-preserving.

The LSH function commonly used in Euclidean Space, which was proposed by Datar [2], is shown as the following:

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

Here, a is a random vector with each dimension independently chosen from Guassian distribution and p is an arbitrary point in D. b is a real number uniformly drawn from the range [0,1]. W is also a real number representing the width of the LSH function. For two points p_1, p_2 and an LSH function h, if $\| p_1, p_2 \| = r$, the probability of $h(p_1) = h(p_2)$ can be computed as follows [2].

$$\begin{aligned} p(r, W) &= Pr[h(p_1) = h(p_2)] \\ &= \int_0^W \frac{1}{r} f_2(\frac{t}{r})(1 - \frac{t}{W})dt \\ &= 2norm(W/r) - 1 - \frac{2}{\sqrt{2\pi}} \frac{r}{W}(1 - e^{-\frac{W^2}{2r^2}}) \end{aligned} \tag{2}$$

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.

Due to the distance-preserving property of LSH, it is rational to use the hash values to estimate the distance between two points. Therefore, if two points have similar hash values, it is believed that they are close to each other with certain confidence. Based on this idea, several approaches have been proposed for c-ANN [2, 4, 5, 12, 19]. However, it is obvious that Equation 1 exhibits poor performance in filtering irrelevant points, as many pairs, which are distant from each other, may share the same hash value under a single hash function as Equation 1. In other words, numerous false positives may be returned. To remove the irrelevant points (i.e. false positives), a compound LSH function $G = (h_1, h_2, \ldots, h_m)$ is employed so as to improve the distinguishing capacity. Note that each element of a compound LSH function, h_i, is randomly selected as defined in Equation 1. Only points sharing all the m hash values with the query point are taken into account as candidate points, as suggested by the basic LSH [2]. However, c-ANN search algorithms should ensure that data points having similar compound hash keys to the query point are taken into account as candidates. Hence, a distance measure over compound hash keys is required. In the following, we propose a novel measure to evaluate the distance between a pair of compound hash keys.

3.2 Distance over Compound Hash Keys

Given a compound LSH function G and two points $p_1, p_2 \in D$, we have compound hash keys $K_1 = G(p_1)$ and $K_2 = G(p_2)$, where both K_1 and K_2 are tuples containing m hash values. Let $k_{1,i}$ (resp. $k_{2,i}$) be the i-th element of K_1 (resp. K_2), that means $k_{1,i} = h_i(p_1)$ and $k_{2,i} = h_i(p_2)$.

Definition 2. ***(Prefix of a Compound Hash Key)*** *Given a point $p \in D$ and its compound hash key $K = G(p) = (k_1, k_2, \ldots, k_m)$. **The l-length prefix of** K, denoted as $pref(K, l)$, consisting of the first l elements of K where $1 \leq l \leq m$, is formally defined as follows.*

$$pref(K, l) = (k_1, k_2, \ldots, k_l) \tag{3}$$

Particularly, we denote $pref(K, 0)$ as $K_\emptyset$, which is actually an empty hash key.

Here, we are inspired by the prefix of a character string and treat a compound hash key as a string of elements. Therefore, its prefix is the substring constituted by its first several elements. For example, for a compound hash key $K = (1, 2, 3, 4)$, $pref(K, 3) = (1, 2, 3)$, $pref(K, 2) = (1, 2)$ and $pref(K, 0) = K_\emptyset$.

Definition 3. ***(Non-prefix Length of Compound Hash Keys)*** *Given two compound hash keys $K_1 = (k_{1,1}, k_{1,2}, \ldots, k_{1,m})$ and $K_2 = (k_{2,1}, k_{2,2}, \ldots, k_{2,m})$, if $pref(K_1, l) = pref(K_2, l)$ and $pref(K_1, l + 1) \neq pref(K_2, l + 1)$, where $0 \leq l < m$, then the **non-prefix length** between K_1 and K_2, denoted as $KL(K_1, K_2)$, is formally defined as follows:*

$$KL(K_1, K_2) = m - l \tag{4}$$

If $pref(K_1, m)$=$pref(K_2, m)$, then $KL(K_1, K_2) = 0$.

A smaller non-prefix distance between two compound hash keys indicates that they share a longer common prefix with each other. For instance, given two compound hash keys $K_1 = (1, 2, 3, 4)$ and $K_2 = (1, 2, 3, 5)$, $KL(K_1, K_2) = 1$ since $pref(K_1, 3) = pref(K_2, 3)$ and $pref(K_1, 4) \neq pref(K_2, 4)$.

Definition 4. ***($(l + 1)$-th Element Distance of Compound Hash Keys)*** *Given two compound hash keys $K_1 = (k_{1,1}, k_{1,2}, \ldots, k_{1,m})$ and $K_2 = (k_{2,1}, k_{2,2}, \ldots, k_{2,m})$, if $KL(K_1, K_2) = m - l$, where $0 \leq l < m$, then the $(l + 1)$**-th element distance** between K_1 and K_2 is defined as the absolute value of the distance between their $(l + 1)$-th elements as follows:*

$$KD(K_1, K_2) = |k_{1,l+1} - k_{2,l+1}| \tag{5}$$

If $l = m$, we denote $KD(K_1, K_2)$ as 0 by default.

Though the notion of non-prefix length can be used to measure the distance between two compound hash keys, we employ the $(l+1)$-th element distance to further distinguish two compound hash keys. Let us consider the following three compound hash keys, $K_1 = (1, 2, 3, 4)$, $K_2 = (1, 2, 3, 5)$ and

$K_3 = (1, 2, 3, 2)$. If only the non-prefix length is applied, we cannot determine which of K_2 and K_3 is more similar to K_1 since $KL(K_1, K_2) = KL(K_1, K_3) = 1$. However, by using the $(l + 1)$-th element distance, it is rational to conclude that K_2 is more similar to K_1 than K_3 according to the fact that $KD(K_1, K_2) < KD(K_1, K_3)$. Thus, for two compound hash keys, we use a linear combination of the non-prefix length and the $(l + 1)$-th element distance to reflect the overall distance. The formal definition of which is shown as follows.

Definition 5. ***(Distance of Compound Hash Keys)*** *Given two compound hash keys K_1 and K_2, the **distance** between them, denoted as $dist(K_1, K_2)$, is defined as follows:*

$$dist(K_1, K_2) = KL(K_1, K_2) + \frac{KD(K_1, K_2)}{C} \tag{6}$$

Here, C is a normalization factor which satisfies for any pair of compound hash keys $K^{'}$ and $K^{''}$, $KD(K^{'}, K^{''}) < C$ holds.

It can easily be proved that the distance measure is a metric, which satisfies nonnegativeness, symmetry and triangle inequality simultaneously. For any pair of compound hash keys, a smaller distance between them indicates that they are more similar to each other. For example, suppose there are three compound hash keys, $K_1 = (1, 2, 3, 4)$, $K_2 = (1, 2, 3, 5)$ and $K_3 = (1, 2, 3, 2)$. Here, C is set to 10, a large enough value. Hence, we have $dist(K_1, K_2) = 1.1$ and $dist(K_1, K_3) = 1.3$. Therefore, K_2 is more similar to K_1 than K_3.

Moreover, there exists a link between the distance measure and the Euclidean distance, which is shown as follows.

Lemma 1. *For two points $p_1, p_2 \in D$ and $\parallel p_1, p_2 \parallel = r$, the distance $dist(G(p_1), G(p_2))$ is less than $m - l + 1$ $(0 \leq l \leq m)$ with probability $[p(r, W)]^l$.*

Proof. *According to Definition 5, if $dist(G(p_1), G(p_2)) < m - l + 1$, we have $KL(G(p_1), G(p_2)) \leq m - l$ and further $G(p_1)$ and $G(p_2)$ at most shares l common prefix. It means that $h_i(p_1) = h_i(p_2)$ holds for $1 \leq i \leq l$. Due to the fact that each LSH function is independently and randomly selected according to Equation 1, the probability is correctly obtained as shown in Equation 7.*

$$\begin{aligned} & Pr[dist(G(p_1), G(p_2)) < m - l + 1] \\ & = \textstyle\prod_{i=1}^{l} Pr[h_i(p_1) = h_i(p_2)] \\ & = [p(r, W)]^l \end{aligned} \tag{7}$$

□

According to Equation 2 and Equation 7, when W and l are fixed, the probability of $dist(G(p_1), G(p_2)) < m - l + 1$ decreases monotonically with the increase of r. For any distance value $m - l + 1$, the smaller $\parallel p_1, p_2 \parallel$ is, the larger the probability of $dist(G(p_1), G(p_2)) < m - l + 1$. Hence, Lemma 1 guarantees that p_1 and p_2 are close to each other when $dist(G(p_1), G(p_2))$ is sufficiently small. Besides, a point with a sufficiently close compound hash key to that of a query according to Equation 6 could be seen as a high-quality candidate for the query.

Notably, LSH functions preserve the distance between points after hashing with a certain probability according to Equation 2. That may cause a number of false negatives: the points which are close to the query point but exhibit a long distance with respect to the hash values. In fact, for most existing methods in ANN search, the basic idea is to verify a set of high-quality candidates so as to find as many true positives as possible with as little cost as possible. Moreover, it is unavoidable for each approach of ANN to lose false negatives.

In line with state-of-the-art approaches in ANN search, false negative is also a big challenge in our approach, especially when we define the distance between compound hash keys using prefix, as described in Definition 5.

In practice, it seems that simply counting the number of common values among m hash values of two points under a compound LSH function makes more sense than our distance measure, which will be proved by experiments in Section 5.3. However, as will be shown in Section 3.3, simply counting the number of collisions cannot be formed as a linear order so as to improve the probability of local distribution of candidates, which can further reduce random I/Os. Moreover, for two compound hash keys, the length of their common prefix can be seen as a lower bound of their total number of common hash values.

According to Lemma 1, the probability $[p(r, W)]^l$ may be very small when l is large enough, which may lead to a large number of false negatives. In order to reduce the loss of false negatives, we use a set $\mathcal{G}$ of $\mathcal{L}$ ($\mathcal{L} > 1$) compound LSH functions, $\mathcal{G} = \{G_1, G_2, \ldots, G_{\mathcal{L}}\}$, to measure the distance among compound hash keys of p_1, p_2 as follows.

$$Dist(\mathcal{G}(p_1), \mathcal{G}(p_2)) = \min_{i=1}^{\mathcal{L}} dist(G_i(p_1), G_i(p_2)) \tag{8}$$

In Equation 8, for any two points p_1 and p_2, there are $\mathcal{L}$ distance values $dist(G_i(p_1), G_i(p_2))$ for $1 \leq i \leq \mathcal{L}$, in respect to $\mathcal{L}$ compound LSH functions and we use the minimum one as the final distance between their corresponding compound hash keys. For a close pair, they may fail to have a small distance under a compound LSH function in $\mathcal{G}$ but they could have small distances under other compound LSH functions in $\mathcal{G}$. It reduces the possibility that close pairs do not have a small distance and hence reduces the number of lost false negatives. Recall that each compound LSH function, $G_i (1 \leq i \leq \mathcal{L})$, is independently and randomly generated. Therefore, the probability that the distance between the corresponding compound hash keys of p_1, p_2 is at most $m - l + 1$ is significantly enlarged as follows.

Lemma 2. *For two points $p_1, p_2 \in D$, $\parallel p_1, p_2 \parallel_2 = r$ and a set $\mathcal{G}$ of $\mathcal{L}$ ($\mathcal{L} > 1$) compound LSH functions, the distance $Dist(\mathcal{G}(p_1), \mathcal{G}(p_2))$ is less than $m - l + 1$ $(0 \leq l \leq m)$ with probability $1 - [1 - p(r, W)^l]^{\mathcal{L}}$.*

Proof. *As each $G_i (1 \leq i \leq \mathcal{L})$ is independently and randomly generated, under which the distance in hash values between p_1 and p_2 is guaranteed by Lemma 1. Then the probability of $Dist(\mathcal{G}(p_1), \mathcal{G}(p_2))$ less than $m - l + 1$ can be computed as follows.*

$$\begin{aligned} & Pr[Dist(\mathcal{G}(p_1), \mathcal{G}(p_2)) < m - l + 1] \\ & = 1 - \textstyle\prod_{i=1}^{\mathcal{L}} (1 - Pr[dist(G_i(p_1), G_i(p_2)) < m - l + 1]) \\ & = 1 - [1 - p(r, W)^l]^{\mathcal{L}} \end{aligned} \tag{9}$$

□

3.3 Linear Order over Compound Hash Keys

To improve the probability for finding candidates within a neighborhood in one single hash table, we introduce a linear order over compound hash keys. For a single compound LSH function, the linear order guarantees that points with close compound hash keys to that of a query according to the distance measure in Equation 6 are stored in consecutive areas. Recall that we assume that there is only one compound LSH function in this subsection for the sake of discussion.

For a data set D and a compound LSH function G, let $\mathcal{K} = \{G(p)\ |p \in D\}$. First, we define an algebra system $\mathcal{K}$ and a binary relation $\leq_G$ for compound hash keys. Then, $< \mathcal{K}, \leq_G >$ is proved to be a linear order set, such that state-of-the-art sorting algorithms (e.g., quick sort, heap sort) can be employed to sort the compound hash keys. Moreover, we prove that for a compound hash key K_i in the set of sorted compound hash keys, the similar compound hash keys to K_i are in its neighborhood.

Definition 6. ***(Relation between Compound Hash Keys)*** *Given two compound hash keys with m elements, K_1 and K_2, we define their relation as follows:*

$$\begin{cases} K_2 <_G K_1 & \text{if } l < m \text{ and } k_{1,l+1} > k_{2,l+1} \\ K_1 =_G K_2 & \text{if } l = m \\ K_1 <_G K_2 & \text{if } l < m \text{ and } k_{1,l+1} < k_{2,l+1} \end{cases}$$

where $l = m - KL(K_1, K_2)$.

We denote a binary relation $K_1 \leq_G K_2$ if and only if $K_1 <_G K_2$ or $K_1 =_G K_2$. We prove that the relation $\leq_G$ on $\mathcal{K}$ is in a linear order in Lemma 3.

Lemma 3. *Compound hash keys in the algebra system $< \mathcal{K}, \leq_G >$ is a linear order set.*

Proof. *For $\forall K_1, K_2, K_3 \in \mathcal{K}$, four properties of a linear order, including reflexivity, antisymmetry, transitivity, and totality, are proved sequentially as follows.*

- *Reflexivity: $K_1 \leq_G K_1$. As $K_1 =_G K_1$ according to the definition of $\leq_G$, $K_1 \leq_G K_1$.*
- *Antisymmetry: if $K_1 \leq_G K_2$ and $K_2 \leq_G K_1$, then $K_1 =_G K_2$. Assume that $K_1 \neq_G K_2$, according to the definition of $\leq_G$, $K_1 <_G K_2$ and $K_2 <_G K_1$. They are contradictory according to Definition 6. Therefore, $K_1 =_G K_2$.*
- *Transitivity, which indicates that if $K_1 \leq_G K_2$ and $K_2 \leq_G K_3$, then $K_1 \leq_G K_3$. Without loss of generality, we have $l_1 = m - KL(K_1, K_2)$ and $l_2 = m - KL(K_2, K_3)$. First, let us consider two special cases listed as follows.*

  (1) *$l_1 = m$. Then, it is concluded that $K_1 =_G K_2$ and thus $K_1 \leq_G K_3$.*

  (2) *$l_2 = m$. Then, it is acquired that $K_2 =_G K_3$ and thus $K_1 \leq_G K_3$.*

  For the other cases where $0 \leq l_1, l_2 < m$, there are 3 cases with respect to the relationship between l_1 and l_2 as follows:

  (1) *$l_1 < l_2$. It is obvious that $pref(K_1, l_1) = pref(K_2, l_1) = pref(K_3, l_1)$ and $k_{1,l_1+1} < k_{2,l_1+1}$, $k_{2,l_1+1} = k_{3,l_1+1}$. Moreover, $k_{1,l_1+1} < k_{3,l_1+1}$. Hence, $K_1 \leq_G K_3$.*

  (2) *$l_1 = l_2$. It is obvious that $pref(K_1, l_1) = pref(K_2, l_1) = pref(K_3, l_1)$ and $k_{1,l_1+1} < k_{2,l_1+1}$, $k_{2,l_1+1} < k_{3,l_1+1}$. Moreover, $k_{1,l_1+1} < k_{3,l_1+1}$. Hence, $K_1 \leq_G K_3$.*

  (3) *$l_1 > l_2$. It is obvious that $pref(K_1, l_2) = pref(K_2, l_2) = pref(K_3, l_2)$ and $k_{1,l_2+1} = k_{2,l_2+1}$, $k_{2,l_2+1} < k_{3,l_2+1}$. Moreover, $k_{1,l_1+1} < k_{3,l_1+1}$. Hence, $K_1 \leq_G K_3$.*

- *Totality, representing that $K_1 \leq_G K_2$ or $K_2 \leq_G K_1$, also holds, which can be easily proved according to Definition 6.*

Therefore, $\leq_G$ is a linear order on $\mathcal{K}$. □

As $\leq_G$ on $\mathcal{K}$ is a linear order according to Lemma 3, existing sorting algorithms can be employed to sort the compound hash keys of all points in D. Recall from the aforementioned discussion, an efficient index structure should store similar points in consecutive space, so that the number of random I/O accesses when performing a query is minimized. Thus, we prove that the new order system over compound hash keys guarantees that the closest compound hash keys to that of a query point q (i.e. $G(q)$) are stored locally.

For simplicity, we denote the set of sorted compound hash keys sequence as $\mathcal{K}'$ and the compound hash key of the query point as K_q. Thus, $\mathcal{K}'$ is monotonically non-decreasing, which means $K_i \leq_G K_j$ if $K_i, K_j \in \mathcal{K}'$ and $i < j$. Without loss of generality, assume $K_q \notin \mathcal{K}'$ and there are two compound hash keys $K_i, K_{i+1} \in \mathcal{K}'$ satisfying $K_i <_G K_q <_G K_{i+1}$ and $1 \leq i < |\mathcal{K}'|$. Intuitively, the closest compound hash key to K_q is either K_i or K_{i+1}, which can be proved by the following two lemmas.

Lemma 4. *Given three compound hash keys with m elements, K_1, K_2 and K, if $K_2 <_G K_1 <_G K$, then $dist(K_1, K) \leq dist(K_2, K)$.*

Proof. *This lemma can be proved by contradiction. Assumed that $dist(K_1, K) > dist(K_2, K)$. For simplicity, let $l_1 = m - KL(K_1, K)$, $d_1 = KD(K_1, K)$, $l_2 = m - KL(K_2, K)$ and $d_2 = KD(K_2, K)$. Here, $0 \leq l_1, l_2 < m$, $d_1, d_2 > 0$ since $K \neq_G K_1$ and $K_2 \neq_G K$. According to Definition 5, there are 2 cases to be taken into account for $dist(K_1, K) > dist(K_2, K)$.*

- *$l_1 = l_2$ and $d_1 > d_2$. According to Definition 3 and Definition 4, $pref(K_1, l_1) = pref(K, l_1) = pref(K_2, l_1)$ and $k_{l_1+1} - k_{1,l_1+1} > k_{l_1+1} - k_{2,l_1+1}$. Further, $k_{1,l_1+1} < k_{2,l_1+1}$. According to Definition 6, $K_1 <_G K_2$, which contradicts the fact $K_2 <_G K_1$.*

- *$l_1 < l_2$. According to Definition 3, $pref(K_1, l_1) = pref(K, l_1) = pref(K_2, l_1)$, $k_{1,l_1+1} < k_{l_1+1}$ and $k_{l_1+1} = k_{2,l_1+1}$. Further, $k_{1,l_1+1} < k_{2,l_1+1}$. According to Definition 6, $K_1 <_G K_2$, which contradicts the fact $K_2 <_G K_1$.*

The conclusion contradicts with the assumption. Therefore, the lemma is proved. □

Lemma 5. *Given three compound hash keys with m elements, K_1, K_2 and K, if $K_2 >_G K_1 >_G K$, then $dist(K_1, K) \leq dist(K_2, K)$.*

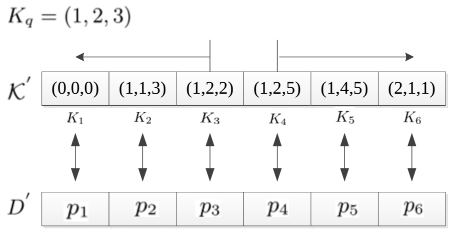


Figure 1: Search strategy

PROOF. *It can be proved in a similar way to Lemma 4.* □

Let D' be the sequence of points after rearranging them in the orders of their corresponding compound hash keys in $\mathcal{K}'$. And, $K_i \in \mathcal{K}'$ corresponds to the point p_i in D'. Assuming that $K_i <_G K_q <_G K_{i+1}$, we can ensure that the closest compound hash key in $\mathcal{K}'$ to K_q is either K_i or K_{i+1}. Moreover, if more close compound hash keys to K_q are required, we can easily explore those adjacent to K_i or K_{i+1}, as illustrated in Figure 1.

As shown in Figure 1, there is a data set with 6 points, $\{p_1, \ldots, p_6\}$ and the set of their corresponding compound hash keys $\{K_1, \ldots, K_6\}$ are sorted in the ascending order. Initially, the set of candidate points $\mathcal{C}$ is $\emptyset$. Obviously, we have $K_3 <_G K_q <_G K_4$ holds. Thus, according to Lemma 4 and 5, the closest compound hash key to K_q is in the set $\psi = \{K_3, K_4\}$. Specifically, K_3 is closer to K_q than K_4 and hence p_3 is added to $\mathcal{C}$. To obtain the second closest compound hash key to K_q, we use the one adjacent to K_3 along the left direction, i.e. K_2, to replace K_3 in ψ. Moreover, it is easy to derive that the second closest compound hash key to K_q is still in ψ. Similarly, after removing K_4 from ψ, K_5, to the right of K_4, is put into ψ. More close compound hash keys will be found in the same way and their corresponding points are put into $\mathcal{C}$.

Finally, we can see that close compound hash keys to K_q are listed in the consecutive areas of $\mathcal{K}'$. According to the orders of seeking close compound hash keys to K_q, the corresponding points in D' can be verified conveniently. When combining the distance measure in Equation 6, linear order significantly improves the probability that candidates are distributed locally.

Notably, we just proved that candidates are distributed locally in a sorted data set with respect to a single compound LSH function. Recall that, to reduce the loss of false negatives, we employ $\mathcal{L}(\mathcal{L} > 1)$ compound LSH functions to construct our index structures. Accordingly, candidates are distributed among $\mathcal{L}$ sorted data sets, which may consume a lot of random I/Os since adjacent points in the sequence of candidates may be located in different sorted data sets. In order to solve the problem, we propose an index structure in the next section.

4. SK-LSH

Actually, data points in a sorted data set are stored in continuous disk pages and each random I/O reads a disk page once. By arranging the candidates' compound hash keys in a single sorted data set according to the distance measure in Equation 8, candidates are accessed and verified in the unit of disk page. First, a disk page contains points with close compound hash keys and it retains the property that candidates similar to the query are stored locally. Second, we select the disk pages containing close points to the query among $\mathcal{L}$ sorted data sets, where the distance measure in Equation 8 still works. Hence, there are two critical problems to solve, including how to organize disk pages and how to measure the distance between a query point and a disk page, in order to determine the orders of disk pages to be accessed. Actually, the two problems correspond to the index strategy and the search strategy respectively. In this section, we present a novel method, namely SK-LSH, to address both problems.

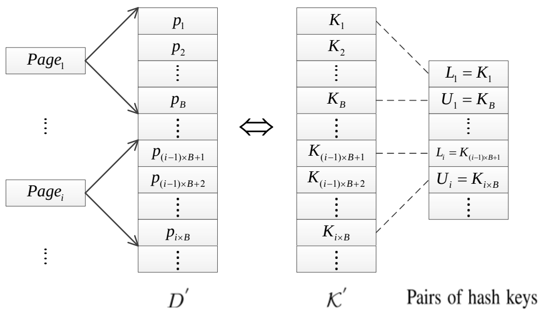


Figure 2: The page structure over sorted data

4.1 Index Strategy

As discussed above, we randomly select a compound LSH function and sort the compound hash keys for $p \in D$ in ascending order. Consequently, the points in D are sorted in the same order correspondingly and stored in consecutive disk pages. Let B be the size of a disk page (i.e. a disk page contains B points at most). We illustrate the data page structure of SK-LSH for a sorted data set in Figure 2.

As can be seen in Figure 2, each data page contains B points at most. The index strategy works as follows. Firstly, the compound hash keys of all points in D are acquired and sorted in the increasing order according to Definition 6, followed by rearranging data points accordingly. Recall that each page, containing B points at most, corresponds to B compound hash keys at most. We denote each page using two representative compound hash keys, the first one and the last one in the data page. In other words, the "lower bound" and the "upper bound" inside a page are used to represent the page. Without loss of generality, $Page_i$ containing B points is referred to as $< K_{(i-1)\times B+1}, K_{i\times B} >$, which are denoted as $< L_i, U_i >$ respectively, as presented in Figure 2.

Using a pair of compound hash keys to represent a disk page, we successfully compress the data set and the compressing ratio is $Bd/2m$, where d is the dimensionality of points in D. In order to organize the representative pairs of all data pages in a sorted data set, we propose to use the popular B$^+$-tree to index them. Its efficiency will be analyzed in Section 4.3.

To sum up, given a compound LSH function and a data set, an index file of SK-LSH consists of two parts, (1) a B$^+$-tree indexing the pairs of compound hash keys for sorted data pages and (2) the sorted data set stored in consecutive data pages. Recall that $\mathcal{L}$ compound LSH functions are employed and thus the index structure of SK-LSH contains $\mathcal{L}$ pairs of B$^+$-trees and sorted data sets.

4.2 Search Strategy

When accessing disk pages during the search process, the order of pages loaded into the memory needs to be decided.

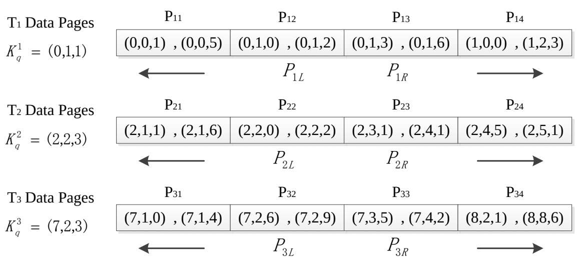


Figure 3: Bi-directional expansion ($m = 3, l = 3$)

To address the issue, we 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 containing B points, let K_q be the compound hash key of q and a pair of compound hash keys $< L, U >$ to denote P. The distance between q and P, denoted as $DIST(q, P)$, is calculated as follows:*

$$DIST(q,P) = \begin{cases} DIST(K_q, L) & \text{if } K_q \le_G L \le_G U \\ 0 & \text{if } L \le_G K_q \le_G U \\ DIST(K_q, U) & \text{if } L \le_G U \le_G K_q \end{cases} \tag{10}$$

As shown in Definition 7, for K_q and $< L, U >$, there are three cases which need to be considered. In the first case, K_q is smaller than all compound hash keys of points in P. So the lower bound of distances between K_q and the compound hash keys of points in P is $dist(K_q, L)$ according to Lemma 5. In the second case where K_q falls into the range between L and U, the distance of K_q and page P is defined as 0. In the third case where K_q is larger than all compound hash keys of points in P, the lower bound is $dist(K_q, U)$ according to Lemma 4.

Based on the above definition, we discuss below how to find the closest page towards q in a sorted data set. Let n be the cardinality of D and $\rho = \lceil \frac{n}{B} \rceil$ be the number of data pages. Besides, we use $\mathbb{P} = \{P_1, P_2, \ldots, P_\rho\}$ to refer to the set of data pages. Hence, a B$^+$-tree is created to index the compound hash keys in $\Theta = \{L_1, U_1, L_2, U_2, \ldots, L_\rho, U_\rho\}$. It is very convenient to find two compound hash keys in Θ, θ_i and θ_{i+1} ($1 \le i < 2\rho - 1$), such that $\theta_i \le K_q \le \theta_{i+1}$ 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 according to Lemma 4 and Lemma 5.

In SK-LSH, we employ $\mathcal{L}$ index files in order to reduce the loss of false negatives. After building a B$^+$-tree on the compound hash keys for each index file, we have $\mathcal{L}$ B$^+$-trees. The key operation for ANN search is to find the next data page to be accessed among $\mathcal{L}$ B$^+$-trees, which can be done by bi-directional expansion at data pages of all B$^+$-trees.

For each B$^+$-tree T_i and $1 \le i \le \mathcal{L}$, we use P_{iL} to denote the closest data page to q in T_i, and P_{iR} the data page immediately succeeding P_{iL}. Figure 3 shows an example where each compound hash key has 3 elements and 3 index files are used.

First, we determine P_{iL} and P_{iR} for T_i. Here, we set C in Equation 6 as 10 for all index files. Taking T_1 as an example, P_{12} has the smallest distance to K_{1q}. Hence, P_{12} is set as P_{1L} and P_{13} as P_{1R}. Similarly, we obtain P_{2L}, P_{2R}, P_{3L} and P_{3R}. It can easily be derived that the page with the smallest distance to its corresponding compound hash key of the query point must be in the set $\Phi = \{P_{1L}, P_{1R}, P_{2L}, P_{2R}, P_{3L}, P_{3R}\}$. To facilitate the discussion, we define two operations on Φ, *extract* and *shift*. Operation $P = extract(\Phi)$ returns the closest data page in Φ from all the B$^+$-trees, and P is meanwhile removed from Φ. $shift(P)$, where $P \in \Phi$, means moving P away from K_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}. Besides, we use the result of $shift(P)$ to replace P in Φ. By repeating *extract* and *shift* operations, close data pages towards the query point will be found from all the B$^+$-trees in sequence.

If the number of data pages verified exceeds the threshold, N_P, specified before the search, SK-LSH terminates the search process. Intuitively, SK-LSH returns superior results with a much less running time than LSB and C2LSH, because it verifies much more points with dramatically less random I/O operations, which is justified and discussed in details in Section 5.

There are $\mathcal{L}$ copies of the data set indexed by $\mathcal{L}$ B$^+$-trees and we access data pages in different index files according to the corresponding distances to the query point. Therefore, it is probable that a candidate point is found in more than one index file. To reduce this unnecessary overhead caused by the duplicate points, we maintain a bitmap for each data point to indicate whether it has been verified, as suggested by A. Andoni and P. Indyk in the manual called *E^2LSH package*[1]. If it has been verified, we just ignore it, which reduces the unnecessary CPU cost. We summarize the whole search process in Algorithm 1.

Algorithm 1 Multi-Tree Search Algorithm

```
Require: $\mathcal{T} = \{T_i | 1 \le i \le \mathcal{L}\}$, $q$, $N_P$;
Ensure: $o$;
 1: for each $i = 1$ to $\mathcal{L}$ do
 2:     Compute the compound hash key of $q$, $K_{iq}$;
 3:     Find out $P_{iL}$ and $P_{iR}$;
 4: end for
 5: repeat
 6:     $P = extract(\Phi)$;
 7:     Verify points in $P$ and update $o$;
 8:     $P' = shift(P)$;
 9:     $\Phi = \Phi \cup P'$;
10: until $|\Phi| \ge N_p$
11: return $o$;
```

4.3 Complexity Analysis

In this subsection, we analyze the space and time complexity of SK-LSH. We first discuss the choice of B, the page size, since it has an influence on our analysis. Intuitively, B should be a relatively large number in order that sorted candidates are distributed locally. In addition, transferring the same amount of data from disk to main memory often requires less time with larger pages than with smaller pages [2]. Hence, a large B significantly improve the efficiency of loading candidates. However, if B is too large, it will weaken the effect of the distance measure in Equation 8,

[1] http://www.mit.edu/~andoni/LSH/.

[2] http://en.wikipedia.org/wiki/Page_(computer_memory)

which is set to reduce the loss of false negatives. Inspired by HashFile [22], we set B as 100 in this paper, indicating that a disk page contains at most 100 points.

The index files of SK-LSH require $\mathcal{L}$ copies of the data set and correspondingly $\mathcal{L}$ B$^+$-trees. A single B$^+$-tree consumes $O(nm/B)$ space, where n is the cardinality of the data set and and m is the number of elements in each compound hash key. Therefore, the total space cost is $O(\mathcal{L}nm/B) + O(\mathcal{L}nd) = O(\mathcal{L}n(m/B + d))$, where d is the dimensionality of data points. Since $m/B \ll d$, the major space requirement comes from the copies of the data sets.

The time complexity consists of two parts, including (1) searching $\mathcal{L}$ B$^+$-trees and (2) loading and verifying points in N_P pages, where N_P is a parameter defined before the ANN search. The cost of (1) exists in exploring $O(\mathcal{L}E)$ pages, where E is the height of a B$^+$-tree. The cost of (2) is proportional to N_P. Notably, the time cost of processing a page is $O(Bd)$. Thus, the total time complexity is $O(Bd(N_P + \mathcal{L}\log_B n))$. As will be justified in Section 5, $\mathcal{L}$ is usually less than 10 in real-world data sets. $\log_B n$ is very small since B is as large as 100. In addition, N_P is usually a small value, which is set as 10 for satisfactory results when comparing with state-of-the-art LSH methods according to our extensive experiments.

4.4 Maintainance of SK-LSH

For an index structure, it is also important to maintain it when the data set is updated. To address this issue, we can simply modify the index files to make the index structure maintainable.

Given a compound LSH function G, we consider the pair of *a point* and *its compound hash key* as *an item* and the compound hash key under G is treated as the key of the item. Instead of indexing only two representative compound hash keys, a B$^+$-tree is built to manage all those items in the modified version. Each internal node only contains keys of their corresponding items while each leaf node includes the whole items. Here, a leaf node corresponds to a data page in the aforementioned SK-LSH. Note that, the size B of a page indicates that a disk page contains at most B points and their corresponding compound hash keys. Since all the data points are stored in the leaves, there are no additional sorted data sets in the modified SK-LSH.

The search process could be performed more directly in the modified B$^+$-trees and the updating of data can also be performed easily. However, a modified B$^+$-tree requires more space because (1) it stores all compound hash keys for each point and (2) the number of points in each leaf node is between $\lfloor B/2 \rfloor$ and B due to the property of B$^+$-trees. However, the increase of space requirement is limited since each leaf node contains at least $\lfloor B/2 \rfloor$ points. The space complexity of modified SK-LSH is $O(\mathcal{L}n(d+m))$.

In addition, the modified index structure will decrease the efficiency and accuracy of SK-LSH slightly and the decreasing effect is also limited. Due to the fact that the number of points in a leaf node is in the range $[\lfloor B/2 \rfloor, B]$, the accuracy of returned results by modified SK-LSH is worse than SK-LSH when accessing the same number of data pages or leaf nodes.

5. EXPERIMENTAL RESULTS

In this section, we investigate the performance of SK-LSH and compare it with state-of-the-art techniques [4, 14, 19] in four multimedia data sets, which are described as follows.

5.1 Set Up

5.1.1 Data Sets

Corel [3]. It consists of 68,040 32-dimensional color histograms from the same number of images.

Aerial [4]. It contains 275,465 60-dimensional texture vectors.

Audio[5]. There are 54,387 192-dimensional vectors in this data set, as is extracted from the LDC SWITCHBOARD-1 collection.

Sift[6]. ANN_SIFT1M consists of three parts, 1,000,000 base vectors, 100,000 learn vectors and 10,000 query vectors, and the dimensionality of each vector is 128.

For each data set, we randomly select 50 points to form the query set and results are averaged.

5.1.2 Performance Measures

We employ three different measures to evaluate the performance of SK-LSH.

- *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 with respect to q, an approach for ANN search 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 with respect to q is computed as

$$ratio(q) = \frac{1}{k}\sum_{i=1}^{k} \frac{\| o_i, q \|}{\| o_i^*, q \|} \tag{11}$$

  In the following experiment, we use the mean of $ratio(q)$ over the query set.

- Average Response Time (ART). The time cost mainly consists of two parts, the searching in $\mathcal{L}$ B$^+$-trees to find the closest data pages to the query point and verifying points to find NNs among $\mathcal{L}$ index files. In SK-LSH, much fewer data pages are necessary to access than LSB and C2LSH while there are indeed more distance computations than LSB and C2LSH. On the other hand, the index structure of CK-Means is very small in size but its CPU cost is considerable. Hence, to make a fair comparison, we use the average response time to evaluate the performance of ANN search. Here, we use t_i to denote the time cost for the i-th query point and n_q the cardinality of the query set.

$$ART = \frac{1}{n_q}\sum_{i=1}^{n_q} t_i \tag{12}$$

- I/O cost (I/O). The efficiency of LSB and C2LSH are sensitive to I/O cost, but they cannot make full use of each random I/O to find enough candidate points. However, SK-LSH successfully overcomes this issue and achieves a much better performance in accuracy and efficiency. Here, we count the number of random I/Os during the search process and the result is denoted as I/O.

[3]http://kdd.ics.uci.edu/databases/CorelFeatures/

[4]http://vision.ece.ucsb.edu/data-sets

[5]http://www.cs.princeton.edu/cass/audio.tar.gz

[6]http://corpus-texmex.irisa.fr/

- Space Requirement. The space requirement of a LSH scheme is the space occupied by its corresponding index structures.

5.1.3 Compared Methods

To demonstrate the superiority of SK-LSH, we compare two categories of methods in this paper: LSH based and PQ based methods. We compare SK-LSH with the state-of-the-art LSH based methods in all of the four performance measures. Due to the space limitation, we only compare SK-LSH and CK-Means, the state-of-the-art PQ based method, in *ratio* and *ART*. Moreover, the cost of PQ based methods is dominated by its CPU cost instead of I/O cost.

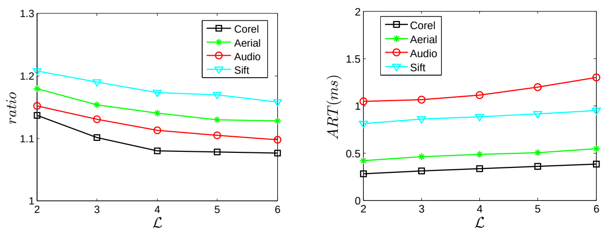
5.2 Effect of Model Parameters

In SK-LSH, there are several parameters which may affect the performance, including the width W for each LSH function, the number m of LSH functions in a compound LSH function, the number $\mathcal{L}$ of index files and the number N_P of data pages accessed during the search process. We will tune these parameters to evaluate the performance of SK-LSH in sequence. By default, in the following experiments we set the number of returned neighbors k as 100 and the size of a disk page B as 100.

Effect of W and m. W and m are two fundamental parameters for constructing an index file. We tune these two parameters for a single index file by fixing $\mathcal{L}$ as 1 and N_P as 10. A group of experiments are conducted to evaluate the performance of SK-LSH under different W and m. The number N_P of data pages accessed is fixed and the time cost will just vary slightly for different settings. Therefore, we are only concerned with the precision of returned neighbors measured by *ratio*. The results are presented in Figure 4.

Generally, the choices of W and m do co-affect the accuracy of returned neighbors according to Figure 4. When W is small, *ratio* does not vary much, which is caused by our randomly selected LSH functions. In **Corel**, when W is 0.2, the smallest value, *ratio* stays around 1.35. Similar phenomena can be found in other data sets. That is because fewer LSH functions with smaller m can achieve an utterly perfect distinguishing capacity and adding more LSH functions will not make any improvements. By contrast, it is easily concluded that if we want to obtain good distinguishing capacity with larger W, m should be large enough. That is also supported by Figure 4. For instance, in Figure 4(b), when W is set to 50, the largest value, *ratio* is more than 1.3 with $m = 10$. In contrast, *ratio* stays around 1.25 as m exceeds 20. Notably, *ratio* just fluctuates slightly when m is large enough for different W's. As shown in Figure 4(b), when m exceeds 20, *ratio* remains stable by varying W. The reason is simple that concatenating more LSH functions is helpless to the distinguishing capacity of a compound LSH function when m is large enough.

By comparing the stable part of *ratios* for different W's, we find that *ratio* is very large if m is small enough. For example, in **Audio**, when m is set as the smallest value, 1, the stable value of *ratio* is obviously worse than that of other choices of W's. As is known, when W is very small, it is highly probable that close pairs will fall into different hash buckets. Our distance measure for compound hash keys is sensitive to such issues, which leads to the missing of candidate points. On the other hand, if W is too large, more LSH functions need to be employed to acquire a good



(a) *ratio* (b) *ART*

Figure 5: Effect of $\mathcal{L}$

distinguishing capacity, which increases the time cost for both the construction of index files and the search.

According to Figure 4, we set (W, m) as (0.8, 12) for **Corel**, (40, 30) for **Aerial**, (3, 30) for **Audio** and (1000, 30) for **Sift** in the following experiments, where SK-LSH exhibits the best performance with respect to *ratio*. Notice that m is smaller than the dimensionality of each data set.

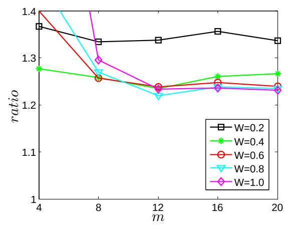
Effect of $\mathcal{L}$. $\mathcal{L}$ is the number of index files which affects the number of false negatives and space consumption. Also, it exhibits impact on the precision of returned neighbors. Intuitively, a larger $\mathcal{L}$ indicates that more information provided by the index files facilitates the accuracy of returned neighbors. To test the effect of $\mathcal{L}$ on *ratio*, we vary $\mathcal{L}$ from 2 to 6 with W and m fixed as above. N_P is fixed to 10 as above. Moreover, we take the response time into account since $\mathcal{L}$ affects time cost of the search process in $\mathcal{L}$ B^+-trees. The results are shown in Figure 5.

As is expected, *ratio* decreases when $\mathcal{L}$ ranges from 2 to 6 in all the data sets as shown in Figure 5(a). This is consistent with our intuitive study since more index files improve the quality of data pages accessed. Also, it justifies that our distance measure proposed in Section 3 is effective. Moreover, as $\mathcal{L}$ increases, the decreasing speed of that decreases showing a submodular pattern [13]. It means increasing the number of index files contributes little to the accuracy of returned neighbors when $\mathcal{L}$ is large enough. Especially, when $\mathcal{L}$ exceeds 4, *ratio* remains relatively unchanged for all those data sets.

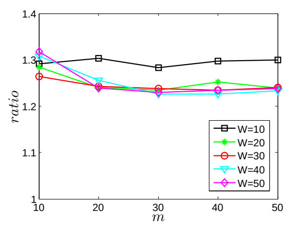
As to the response time, increasing $\mathcal{L}$ leads to more time cost for searching in B^+-trees. The height of a B^+-tree is very small in our experiment, 2 for all the data sets. Moreover, for a page in a B^+-tree, the closest data page to a query point can be found by a binary search since the compound hash keys are stored in increasing order. Thus, it only takes very little time to search in a B^+-tree. Figure 5(b) shows that *ART* does not increase significantly as $\mathcal{L}$ goes up.

The space requirement of SK-LSH is proportional to the value of $\mathcal{L}$. As the performances of SK-LSH in all the data sets with respect to both *ratio* and *ART* are satisfactory and do not vary much if $\mathcal{L}$ is greater than 3, we set $\mathcal{L}$ as 3 in the following experiment unless otherwise specified. In fact, the space requirement of SK-LSH with $\mathcal{L} = 3$ is less than state-of-the-art LSH methods (e.g. LSB and C2LSH), which is shown in Table 1 and will be discussed in detail in Section 5.4.

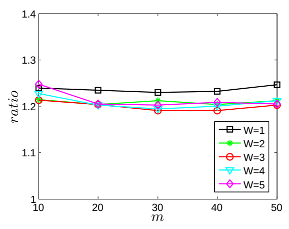
Effect of N_P. The number N_P of data pages accessed for a query directly influence the accuracy of returned neighbors and the response time. Intuitively, more data pages are accessed, more precise neighbors are returned and more time is consumed during the search process. We vary the values for N_P from 5 to 30 at step 5, and the results are shown in



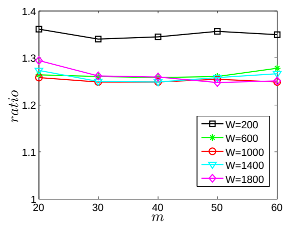
(a) **Corel**



(b) **Aerial**

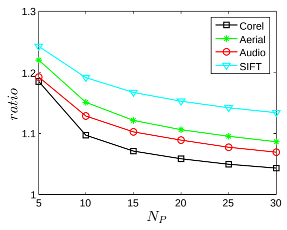


(c) **Audio**

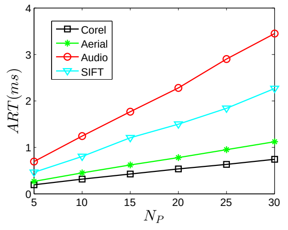


(d) **Sift**

Figure 4: Effect of W and m

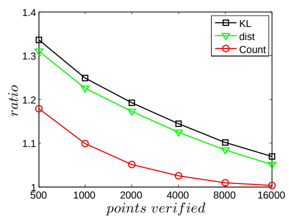


(a) *ratio*

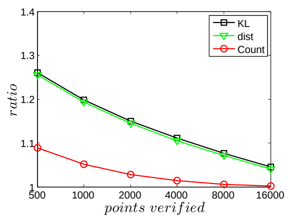


(b) *ART*

Figure 6: Effect of N_P



(a) **Corel**



(b) **Audio**

Figure 7: Effect of distance measures

Figure 6.

As expected, *ratio* keeps on decreasing in all the data sets when N_P increases. As to *ART*, it nearly increases linearly as N_P increases for different data sets. Notably, the increasing rate of *ART* is correlated to the dimensionality of the data set. As can be seen in Figure 6(b), the 192-dimensional **Audio** has the longest response time with the same number N_P whereas the 32-dimensional Color has the shortest response time. It is because the main cost of the search exists in two parts: the exploration in $\mathcal{L}$ B$^+$-trees and dealing with N_P data pages. Both of those two parts are linearly related to the dimensionality of the data set. In addition, due to our strategy of determining the size of a page, the response time for loading a page into the main memory also differs for different data sets.

5.3 Effect of Distance Measures

In this subsection, we focus on the different distance measures for the compound hash keys. There are three distance measures for compound hash keys, namely KL in Definition 3, *dist* in Definition 5 and *Count*, which simply counts the number of common hash values between two compound hash keys. Here, we observe the effects of different measures on the accuracy of returned results by varying the number of points verified. Notably, there is only one compound hash function used. Due to the space limit, we only present the results performed on **Corel** with the smallest dimensionality and **Audio** with the largest dimensionality in Figure 7.

When verifying the same number of points, *Count* guarantees much more accurate results than both KL and *dist* while *dist* has slightly better results than KL since *dist* contains more information (i.e. KD) than KL. As previously mentioned, there is no linear order for *Count* and hence it could not use the efficient index structure in this paper. It can also be seen that differences between *dist* and KL decreases as the dimensionality increases.

5.4 Comparison with LSB and C2LSH

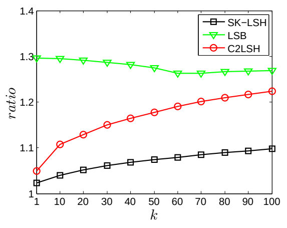
In this part we compare SK-LSH with LSB and C2LSH on the four data sets. Dimensionality ranges from 32 to 192. To make a fair competition, we set the page size as $B = 100$ points for all three LSH methods. We set parameters in SK-LSH according to the above discussion and set N_P as 10. To make a complete comparison, we vary k, the number of returned neighbors, in the set $\{1, 10, 20, 30, ..., 100\}$.

The comparisons of *ratio* on different data sets are presented in Figure 8. Obviously, SK-LSH significantly outperforms LSB and C2LSH in *ratio*, which reflects the precision of returned neighbors, and C2LSH overmatches LSB in most cases. The curves representing *ratio* of SK-LSH and C2LSH rise as k increases on all the four data sets, but that of C2LSH displays a fiercer growing trend. As the curves referring to *ratio* of LSB, those on **Corel**, **Aerial** and **Audio** keeps relatively stable while that on **Sift** in Figure 8(d) shows a gradually increasing trend.

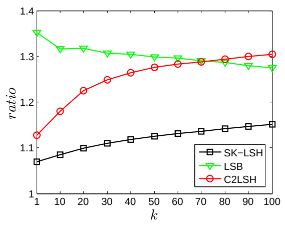
The results on ART are shown in Figure 9. SK-LSH obviously exhibits the best performance comparing with its competitors in *ART*. Especially, in **Sift**, SK-LSH consumes less time than C2LSH by more than an order of magnitude and LSB by two orders of magnitude. In **Audio**, with dimensionality as large as 192, the efficiency of SK-LSH obviously degrades, which indicates that the time cost of SK-LSH is sensitive to the dimensionality of the data set. As discussed in Secion 4.3, when the dimensionality is high, SK-LSH has to spend more time to deal with each page necessary to be accessed. C2LSH consumes significantly less time than LSB in all the data sets.

From Figure 8 and Figure 9, it is obvious that SK-LSH returns *more precise neighbors* than LSB and C2LSH with *the least response time*, which demonstrates the superiority of SK-LSH over LSB and C2LSH.

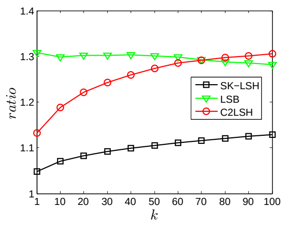
The results of I/O cost are shown in Figure 10. SK-LSH consumes the least I/O cost on all the four data sets. SK-LSH reduces the I/O cost by an order of magnitude compared with the other two approaches on **Aerial** and **Sift**. Notably, the I/O cost of SK-LSH keeps unchanged for different k's and data sets, and this is caused by the fact that the search process of SK-LSH actually consists of two part-



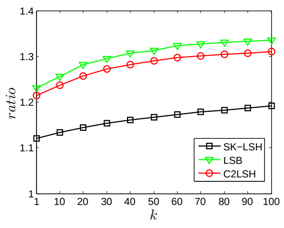
(a) **Corel**



(b) **Aerial**

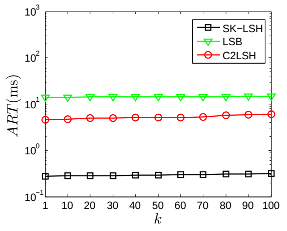


(c) **Audio**

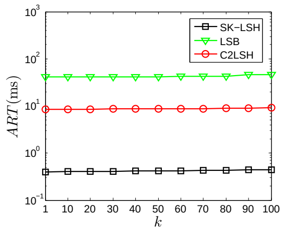


(d) **Sift**

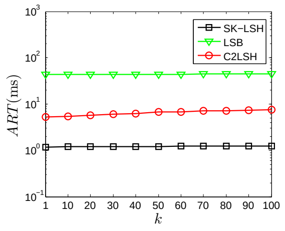
Figure 8: Comparisons with LSB and C2LSH on ***ratio***



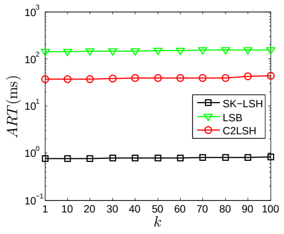
(a) **Corel**



(b) **Aerial**



(c) **Audio**



(d) **Sift**

Figure 9: Comparisons with LSB and C2LSH on ***ART***

Table 1: Comparisons of Space Requirement

Methods		LSB	C2LSH	SK-LSH
Corel	Overall Space Requirement	454MB	159MB	26MB
	Number of Hash Structures	26	208	3
Aerial	Overall Space Requirement	6.1GB	261MB	194MB
	Number of Hash Structures	53	230	3
Audio	Overall Space Requirement	1.5GB	121MB	120MB
	Number of Hash Structures	24	205	3
Sift	Overall Space Requirement	80.6GB	1.09GB	384MB
	Number of Hash Structures	100	250	3

s, i.e. (1) exploration among $\mathcal{L}$ B$^+$-trees and (2) accessing N_P data pages. For each B$^+$-tree, only E, the height of a B$^+$-tree, pages are necessary to be accessed, but E is a very small value according to the analysis in Section 4.3. In our experiments, $E = 2$ for all the four data sets, and thus the I/O cost of SK-LSH is fixed to 16. The I/O cost of LSB and C2LSH are sensitive to the size of the data set. Obviously, both of them consume more I/O cost when dealing with **Sift** than the other three data sets. The I/O cost of C2LSH keeps relatively stable on different data sets while that of LSB varies with the size of the data set. However, the I/O cost of C2LSH increases linearly with the increase of k while that of LSB does not vary with k. This is due to their different terminating conditions, as discussed above. The main I/O cost of C2LSH is incurred to access candidate points and each access to a candidate means a random I/O. For LSB, its I/O is paid to find candidate points with the most similar z-orders to that of the query points among a large number of LSB-trees.

Space requirement is also an important indicator for LSH schema, which is closely related to its applicability and s-calability. The comparison of space requirement for all the four data sets are presented in Table 1. As described above, the space required by SK-LSH consists of two parts: $\mathcal{L}$ B$^+$-trees and $\mathcal{L}$ copies of the data set. The data for SK-LSH in Table 1 takes both two parts into account. SK-LSH only maintains three hash structures, and hence it has the least space requirement.

5.5 Comparisons with CK-Means

In this part, we compare SK-LSH with CK-Means [14], the state-of-the-art method based on PQ. Due to the space limitation, we only present comparisons of *ratio* and *ART* on **Aerial** and **Sift**. Note that CK-Means is a memory-resident index method. To make a fair comparison, we set $\mathcal{L} = 10$ and $N_p = 40$ for SK-LSH. The number of bits for each code is fixed as 64 in CK-Means, as suggested in [14].

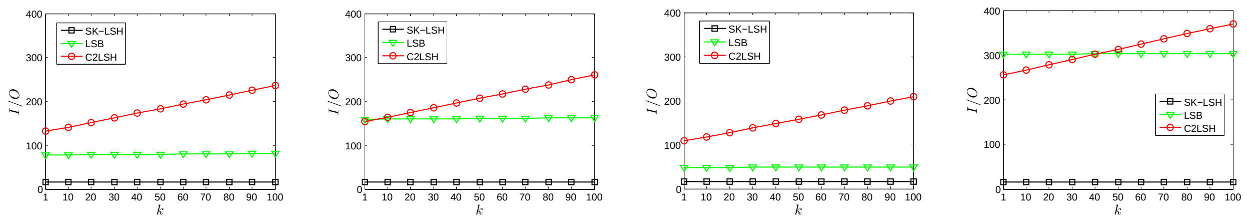
The results of ratio are shown in Figure 11. SK-LSH and CK-Means return results with similar accuracy. Notably, *ratio* of SK-LSH increases with the increasing of k, while that of CK-Means presents an opposite trend. When k is small, SK-LSH returns better results than CK-Means.

From Figure 12, we can see that SK-LSH costs far less time than CK-Means in the two data sets. The main time cost of SK-LSH is on loading and verifying N_p data points, which is sensitive to the dimensionality of the data set but robust to the size of the data set. On the other hand, CK-Means needs to compute the AQD values between all codes and that of the query point. Those costs are very sensitive to the size of the data set but robust to the dimensionality of the data set, given the fixed code length. Since the size of **Aerial** and **Sift** are relatively large, the efficiency of SK-LSH outperforms CK-Means on *ART* by almost an order of magnitude. SK-LSH requires considerably more space than CK-Means because the index file of CK-Means mainly contains 64-bit codes for each point while SK-LSH needs spaces for $\mathcal{L}$ data sets and B^+-trees.

In short, SK-LSH better solves the scalability issue than CK-Means, with more space required.

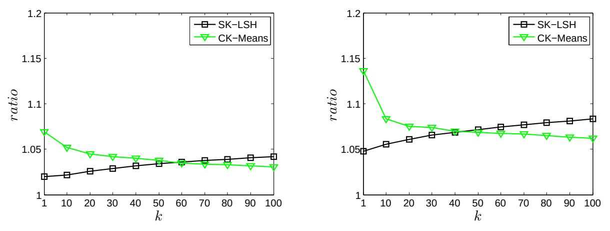
6. CONCLUSION

In this paper, we propose a novel index structure, SK-LSH, to address ANN search based on the popular LSH techniques. To speed up the search process, we introduce a new distance measure for compound hash keys and define a novel relation between them, which makes it possible for close points to be stored locally. Hence, during the search process, we can find sufficiently accurate neighbors by accessing only limited data pages. Extensive experiments conducted over four real-life data sets (varying from small dimension



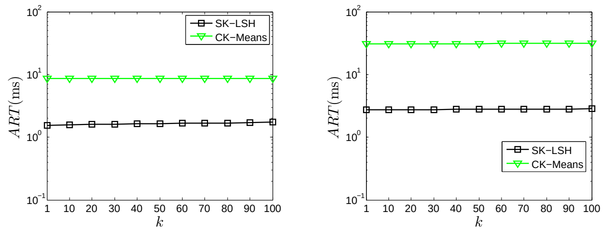
(a) **Corel** (b) **Aerial** (c) **Audio** (d) **Sift**

Figure 10: Comparisons with LSB and C2LSH on I/O cost



(a) **Aerial** (b) **Sift**

Figure 11: CK-Means v.s. SK-LSH on ***ratio***



(a) **Aerial** (b) **Sift**

Figure 12: CK-Means v.s. SK-LSH on ***ART***

to large dimension) demonstrate the superiority of SK-LSH over the state-of-the-art LSH approaches, LSB and C2LSH, and the state-of-the-art non-LSH approach, CK-Means, e-specially with respect to the time cost. In the future, we will explore the applications of the distance measure and the linear order relationship proposed in this paper in other research fields such as large-scale image retrieval.

7. ACKNOWLEDGMENTS

Yingfan Liu, Jiangtao Cui and Hui Li are supported by the National Natural Science Foundation of China (Grant No.61173089 and 61202179) and by SRF for ROCS, SEM. Zi Huang and Heng Tao Shen are supported by ARC Discovery Projects DP110103871 and DP130103252.

8. REFERENCES

[1] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. In *VLDB*, pages 28–39, 1996.

[2] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SoCG*, pages 253–262, 2004.

[3] R. Datta, D. Joshi, J. Li, and J. Z. Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Computing Surveys*, 40(2):1–60, 2008.

[4] J. Gan, J. Feng, Q. Fang, and W. Ng. Locality sensitive hashing scheme based on dyanmic collision counting. In *SIGMOD*, pages 541–552, 2012.

[5] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999.

[6] J.-P. Heo, Y. Lee, J. He, S.-F. Chang, and S.-E. Yoon. Spherical hashing. In *CVPR*, pages 2957–2964, 2012.

[7] Z. Huang, H. T. Shen, J. Shao, S. M. Rüger, and X. Zhou. Locality condensation: a new dimensionality reduction method for image retrieval. In *ACM Multimedia*, pages 219–228, 2008.

[8] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998.

[9] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE TPAMI*, 33(1):117–128, 2011.

[10] A. Joly and O. Buisson. A posteriori multi-probe locality sensitive hashing. In *ACM Multimedia*, pages 209–218, 2008.

[11] N. Katayama and S. Satoh. The sr-tree: an index structure for high-dimensional nearest neighbor queries. In *SIGMOD*, pages 369 – 380, 1997.

[12] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe lsh: efficient indexing for high-dimensional similarity search. In *VLDB*, pages 950–961, 2007.

[13] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions. i. *Math. Programming*, 14(3):265–294, 1978.

[14] M. Norouzi and D. Fleet. Cartesian k-means. In *CVPR*, 2013.

[15] R. Panigrahy. Entropy based nearest neighbor search in high dimensions. In *SODA*, pages 1186 – 1195, 2006.

[16] H. T. Shen, B. C. Ooi, , Z. Huang, and X. Zhou. Towards effective indexing for very large video sequence database. In *SIGMOD*, pages 730–741, 2005.

[17] H. T. Shen, X. Zhou, and A. Zhou. An adaptive and dynamic dimensionality reduction method for high-dimensional indexing. *The VLDB Journal*, 16(2):219–234, 2007.

[18] J. Song, Y. Yang, Y. Yang, Z. Huang, and H. T. Shen. Inter-media hashing for large-scale retrieval from heterogenous data sources. In *SIGMOD*, 2013.

[19] Y. Tao, K. Yi, C. Sheng, and P. Kalnis. Quality and efficiency in high dimensional nearest neighbor search. In *SIGMOD*, pages 563–576, 2009.

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

[21] D. A. White and R. Jamesh. Similarity indexing with the ss-tree. In *ICDE*, pages 516 – 523, 1996.

[22] D. Zhang, D. Agrawal, G. Chen, and A. K. H. Tung. Hashfile: An efficient index structure for multimedia data. In *ICDE*, pages 1103–1114, 2011.