Knowledge and Information Systems manuscript No.
(will be inserted by the editor)

Indexing Expensive Functions for Efficient Multi-dimensional Similarity Search

Hanxiong Chen · Jianquan Liu · Kazutaka Furuse · Jeffrey Xu Yu · Nobuo Ohbo

Received: Oct 30, 2009 / Revised: Mar 30, 2010 / Accepted: Apr 24, 2010

Abstract Similarity search is important in information retrieval applications where objects are usually represented as vectors of high dimensionality. This leads to the increasing need for supporting the indexing of high dimensional data. On the other hand, indexing structures based on space partitioning are powerless because of the well-known "curse of dimensionality". Linear scan of the data with approximation is more efficient in the high dimensional similarity search. However, approaches so far have concentrated on reducing I/O, and ignored the computation cost. For an expensive distance function such as L_p norm with fractional p, the computation cost becomes the bottleneck. We propose a new technique to address expensive distance functions by "indexing the function" by pre-computing some key values of the function once. Then, the values are used to develop the upper/lower bounds of the distance between a data vector and the query vector. The technique is extremely efficient since it avoids most of the distance function computations; moreover, it does not involve any extra secondary storage because no index is constructed and stored. The efficiency is confirmed by cost analysis, as well as experiments on synthetic and real data.

Keywords Similarity search · High dimensional space · Function index

1 Introduction

Implementing efficient similarity search mechanisms for high dimensional data sets is one of the important research topics in the field of data engineering, and has been well studied in recent years [1,2,3,4,5,6]. The difficulties of this topic mainly arise from the well-known property called the "curse of dimensionality". In high dimensional spaces, it is observed that hypersphere covering results of a nearest neighbor query tend to have huge radius.

Hanxiong Chen · Jianquan Liu · Kazutaka Furuse · Nobuo Ohbo
Department of Computer Science, Graduate School of Systems and Information Engineering,
University of Tsukuba, 1-1-1 Tennodai, Tsukuba-shi, Ibaraki-ken, 305-8577, Japan
E-mail: chx@cs.tsukuba.ac.jp, ljq@dblab.is.tsukuba.ac.jp,
furuse@cs.tsukuba.ac.jp, ohbo@cs.tsukuba.ac.jp

Jeffrey Xu Yu
Department of Systems Engineering & Engineering Management, Chinese University of Hong Kong, China
E-mail: yu@se.cuhk.edu.hk

Because of this property, space partitioning and data partitioning techniques become worse than the simple sequential scan [7]. For this reason, recent research attention is mainly paid to improving the performance of the sequential scan. Such techniques include VA-file (or vector approximation file) and its variations [1, 6, 8, 9].

Indeed, the dimensionality curse problem has been dealt with by employing appropriate dimensionality reduction techniques. Some approaches surveyed in [10] can be considered as the same category. SEM-tree [11] treats sequence as vector, and approaches the search efficiency by dimensionality reduction. Earlier researches such as FastMap [12] is also a feature selection technique which can be applied to metric data as well. Classical PCA [13] and SVD [14] are well known. However, dimensionality reduction techniques are impossible or at least not feasible for huge data, for data that the target dimensions are not clear, and for dynamic data like stream.

Another research topic of the similarity search is what distance metric should be used [15, 16, 17, 18]. For data vector sets, L_p norm (Minkowski metric) is generally used in various applications. The parameter p refers to the degree of power, and mainly used are $p = 1$ (Manhattan metric) and $p = 2$ (Euclidean metric). In [15], it is argued that the value of p is sensitive to meaningfulness in high dimensional spaces, and a smaller value of p is preferable. It is also mentioned that fractional values less than 1 are rather more effective than the cases of $p = 1$ and $p = 2$. However, using fractional p makes the distance calculation costly, and it considerably decreases the performance of the similarity search. This is because we need time consuming numerical computations for calculating pth powers.

In this paper, we propose an efficient technique for the similarity search in feature vector space with "expensive" distance functions. The motivation comes from the observation on our first experiment (Figure 10) that the computation cost of the functions affects the performance as significant as the I/O cost. Our approach is based on the idea of "indexing the costly distance function". Some approximated values of pth powers which can be used to calculate the distance between vectors are computed in advance, and they are used to obtain upper and lower bounds in the course of the similarity search procedure. The technique is extremely efficient since we do not have to perform numerical computations for every search. We demonstrate the efficiency of the proposed technique by both analysis and experiments performed on synthetic and real data sets. In [19], an effort is made in avoiding distance calculation by buffering the query (result) history. We do not believe that it is index-free because query results take secondary storage. Further, the management of the query result buffer affects the search efficiency significantly. The main contribution of [20] is to merge the lower dimensional method (R-tree) and the high dimensional method (VA file), but it does not aim at improving the efficiency itself of the original R-tree and VA file.

Integrating with the basic idea of *function index* in [21] the main contributions of this papers are as follows.

- we discover that the computational cost of the distance function is also a bottle neck in the multi-dimensional search, while traditional researches concentrated on the space-partition based index.
- we propose a new scheme that indexes the function values which are calculated on the fly, and thus indeed require no extra secondary storage.
- we design algorithms which can be applied to both range queries and k-NN queries. Our algorithms are extremely efficient for high dimensional search with an expensive function. They are effective enough for relatively lower dimensional search with a normal distance function such as Euclidean distance. The algorithms are flexibly applicable to both lower and high dimensional searches.

This paper is organized as follows. In the next section, we explain the necessity of expensive distance functions, by recovering some arguments on the behavior of non-Euclidean distance metrics in high dimensional space. We then clarify the necessity of the sequential scan and describe the neighbor search problems. Following this, we explain the motivation of our approach with a running example in Section 3. Then, Section 4 describes the proposed method and its performance analysis. In Section 5, we present our empirical results. Section 6 provides a summary and conclusions.

2 Background and Problem Description

Table 1 Notations and Basic Definitions

V, D	vector database, $V \subseteq [0,1)^D$, D: number of dimensions
N	the number of vectors in V, that is $N = \|V\|$
d	subscription range over dimensions, $d \in \{1, 2, \ldots, D\}$
v_i	ith data vector, $v_i \in V$, sometimes be omitted by v
$v.x_d$	v's coordinate value of dth dimension. $v.x_d \in [0,\ 1)$
q	a query vector, $q \in [0,1)^D$
δ_d	the difference between q and v on dth dimension, $\|q.x_d - v.x_d\|$
L_p	L_p-norm distance function. $L_p(q,v) = \sqrt[p]{\sum_{d=1}^{D} \delta_d^p}$
$l_b(v), u_b(v)$	lower & upper bounds of v, respectively: $l_b(v) \leq L_p^p(q,v) < u_b(v)$
B	parameter: the number of *knot*s dividing [0,1)

In this section, we clarify the necessity of expensive distance function by the example of non-Euclidean Distance. We then explain the superiority of the sequential scan over space-partitioning methods in the high dimensional similarity search. Finally we define the problem that we want to solve formally. In additional the notations throughout this paper are summarized in Table 1.

In [16], Beyer et. al. gave the following theorem which shows that the difference between the maximum and minimum distance becomes less significant as the dimensionality increases. This means that in the high dimensional application, it is hard to disseminate the "nearness" among different points. In such cases, all data points may be finally clustered to a single group. Therefore, any spatial index based on space partitioning will be powerless.

Theorem 1 (Adapted for L_p metric) *If* $\lim_{D\to\infty} var\left(\frac{||X_D||_p}{E[||X_D||_p]}\right) = 0$, *then* $\frac{max\{||X_D||_p\} - min\{||X_D||_p\}}{min\{||X_D||_p\}} \rightarrow_p 0$. *Where* $E[X]$ *and* $var(X)$ *express the expected value and the variance of the variable* X *respectively.* $||X_D||_p$ *means the* L_p*-norm of* X *(againsts the origin) in the* D*-dimensional space.*

On the other hand Aggarwal et. al. figured out another result as follows.

Theorem 2 *([15]) Let* $\mathcal{F}$ *be the uniform distribution of* N *points and* $p = 1/l$ *for some integer l. Then,* $C \cdot \sqrt{\frac{1}{2 \cdot p + 1}} \leq \lim_{D\to\infty} \left(\frac{max\{||X_D||_p\} - min\{||X_D||_p\}}{min\{||X_D||_p\}}\right)\sqrt{D} \leq C \cdot (N-1) \cdot \sqrt{\frac{1}{2 \cdot p + 1}}$ *for some constant C.*

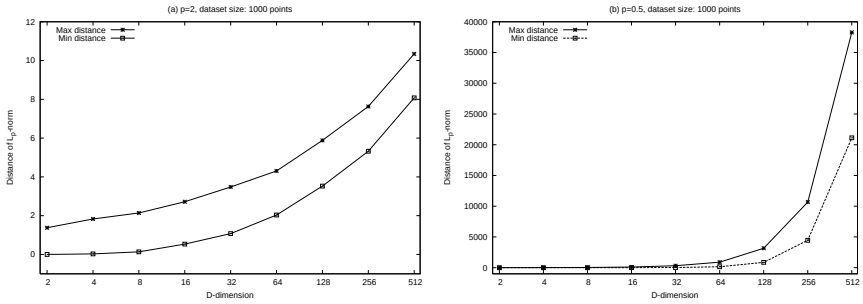


Fig. 1 Change of the maximum and minimum distance with different dimensionalities.

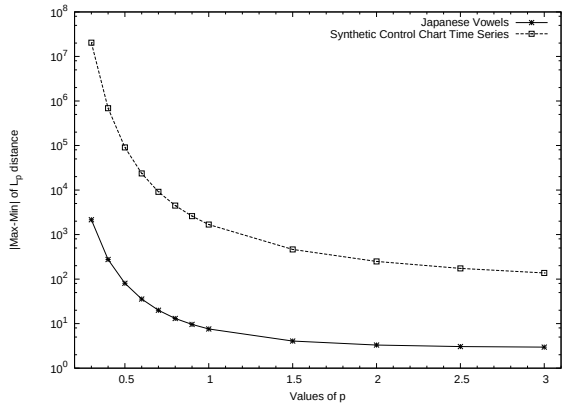


Fig. 2 Change of the maximum and minimum with different p's.

By this, they argue that though the discrimination affected by dimensionality negatively, a smaller p helps to affect it positively.

We confirm the result by a real dataset and a synthetic dataset. Figure 1 (a) shows the confirmation of the first theorem: the maximum and minimum distance increase almost in the same rate for $p = 2$; while in (b) of the figure where $p = 0.5$, the maximum distance increases much faster than the minimum distance as the dimensionality increases, which is the conclusion of the second theorem.

Figure 2 shows yet another result for the real datasets taken from the online UCI KDD Archive [1]. We compare the change against p because the dimensionalities of the two datasets are fixed (12 and 60, respectively). Obviously for both datasets the difference increases when p decreases.

Figure 3 displays some shapes of L_p spaces for different p's. As shown in Figure 2, the difference between the maximum and minimum distance decreases as p enlarges. As the extreme example, the unit sphere of L_∞ is the square and when the dimensionality is very high. The vectors seem to concentrate on the sphere face and make it difficult to distinguish each other. From these spaces we also observe two properties which affect the search result and efficiency.

[1] http://kdd.ics.uci.edu/summary.task.type.html

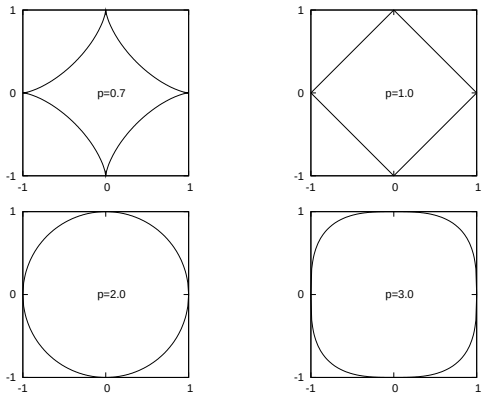


Fig. 3 The *Unit spheres* of different p's. Vectors on the sphere have the same distance from the center and cannot be distinguished from each other.

Property 1 *L_p does not keep distance order for different p.*

In other words, let $p1 \neq p2$, and $w_1,\ v_1,\ w_2,\ v_2$ be four vectors in the same dimensional space. Then $L_{p1}(w_1,\ v_1) > L_{p1}(w_2,\ v_2)$ does not imply $L_{p2}(w_1,\ v_1) > L_{p2}(w_2,\ v_2)$. It needs only an example to prove this property. In the two dimensional space $[0,1)^2$, let $w_1 = w_2 = (0,0), v_1 = (0,1)$, and $v_2 = (0.5, 0.5)$. Then $1 = L_2(w_1,\ v_1) > L_2(w_2,\ v_2) = \sqrt{0.5^2 + 0.5^2}$. On the other hand, $1 = L_{1/2}(w_1,\ v_1) < L_{1/2}(w_2,\ v_2) = (\sqrt{0.5} + \sqrt{0.5})^2$.

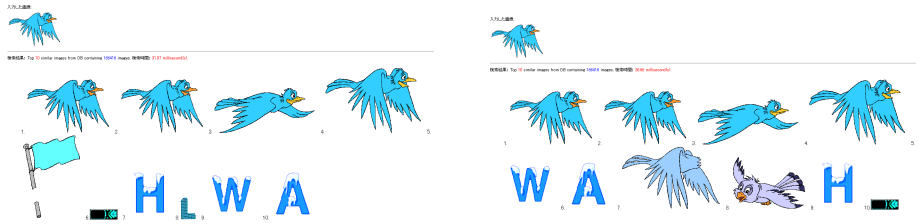


Fig. 4 Image search result for 10-NN on (left) L_2 space and $L_{0.3}$ space, respectively. On top of each figure is the same query image.

This explains our demonstration result of Figure 4. In this example, we searched 10-NN from *Corel GALLERY™ 1,000,000* for p taken 0.3 and 2 respectively. In both cases, the top 3 most similar images are found. But by $p = 0.3$, it also finds two other images (the 6'th and the 7'th) which look more similar than by $p = 2$.

Though L_p for general p leads to a good search result, it brings trouble as well. The following property points to the cases when space partitioning index will be useless.

Property 2 *When $p < 1$, L_p is not metric space.*

Similarly, we need only to show an example that violates any metric conditions. Also in $[0,1)^2$, let $o = (0,0), u = (0,1)$, and $v = (1,1)$. Then $L_{1/2}(o,\ u) + L_{1/2}(u,\ v) = 1 + 1$ is smaller than $L_{1/2}(o,\ v) = (\sqrt{1} + \sqrt{1})^2$ which does not satisfy the triangle inequality.

Consequently, and similar to the argument in [15], a distance function other than the Euclidean one is necessary in high dimensional applications. On the other hand, the computing cost of a L_p is some hundred times more than addition and/or multiplication. Therefore, we would like to propose efficient and effective method that avoids L_p computation.

As for comparison between partition-based indexes and the sequential scan, [1] confirmed the priority of the sequential scan by three conclusions, in terms of performance, complexity and degeneration. More detailedly, it pointed out that for any clustering and partitioning method there is a dimensionality beyond which all blocks are accessed and hence the complexity becomes *O(size-of-DB)*, and therefore the sequential scan performs better.

The problems we are to solve effectively and efficiently in this paper are the *range query* and *k-NN query* which are defined in the following.

Range queries

Given a data vector set V, a vector q and a real number r, the range query finds vectors within distance r from q, that is, $v|v \in V \wedge dist(q, v) \leq r$.

k-NN queries

Given a data vector set V, a vector q and a natural number k, the k-NN query finds the k vectors nearer than other vectors from q. Let the answer of the k-NN query be V_k, then $\forall v \in V_k, v' \in V - V_k, dist(q, v) < dist(q, v')$, $|V_k| \geq k$ and $|V_{k-1}| < k$

3 Motivation by Example

The motivation of our idea is illustrated by examples (Figure 5 and 6). We first explain how the bounds are assembled with Figure 5. It is worth emphasizing that the virtual grid lines are drawn only for understanding, and that there is no necessity to really partition the space. For simplicity, suppose that the space is two dimensions and normalized (that is, $[0, 1)^2$) and that the distance function is L_p with a fractional p ($0 < p < 1$). The first example supposes that the problem is to find answers from a large set of vectors within a distance r away from a given query point $q = (q.x,\ q.y)$. Because the calculation of $(\cdot)^p$ is expensive, our aim is to avoid such calculations as much as possible.

Suppose that the interval [0, 1) is divided into 10 equal blocks by points $t/10,\ t = 0, 1, \ldots, 10$. Preparing $c[t] = (t/10)^p,\ t = 1, \ldots, 9$ costs 9 calculations of p-power ($c[0] = 0$ and $c[10] = 1$ are known easily). These $c[t]$'s are enough to construct the upper and lower bounds for the efficient nearest search for *all* vectors. In other words, though $c[\cdot]$ is one dimensional, it is commonly used for all the dimensions.

The pre-computed $c[\cdot]$ are used to assemble the values of the "knots" of the grid net, for instance, $L_p^p(q, p1) = c[2] + c[2]$. Note that the first $c[2]$ expresses that the distance between q and $p1$ projected on x-axis is 2 knots, while the second $c[2]$ expresses the same distance on y-axis, and they share the same array $c[\cdot]$. Each vector falls in one of the rectangles bounded by 4 corner knots, whose nearest one and farthest one to q represent the lower bound and the upper bound of the distance between v and q, respectively. For vector v of Figure 5 c), the bounds are $L_p(q, p1)$ and $L_p(q, p2)$ because $L_p(q, p1) \leq L_p(q, v) < L_p(q, p2)$. We would like to emphasize that, the grid net in the figure is drawn only for the sake of understanding. It does not represent the partition of the real space, nor the coordinate system. Hence, as

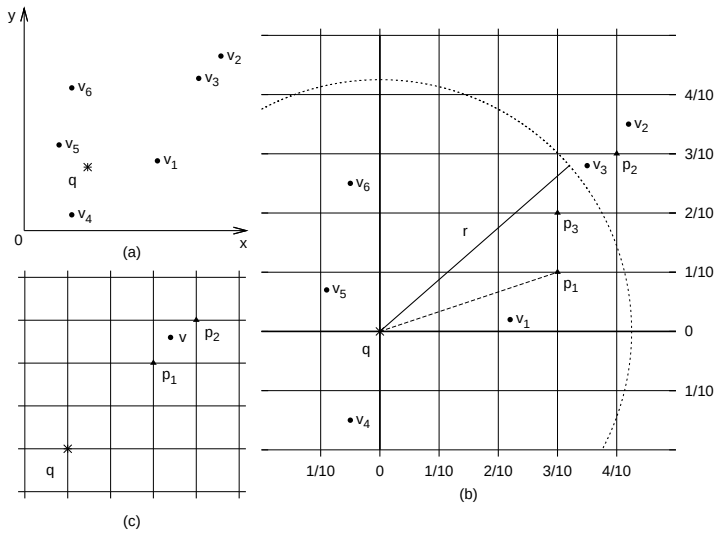


Fig. 5 A 2-dim. example of function index: (a) the data space and an arbitrary query vector; (b) "cover" the data space with a virtual grid net centered at q; and (c) focus on one vector to see how bounds are assembled.

shown in Figure 5 a) and b), it never means that the query point q must locate in the origin, but instead q can be assigned anywhere.

In Figure 5 c), taking a data vector $v = (v.x,\ v.y)$, suppose the task is to find whether $L_p(q, v) \leq r$ holds. Before computing the exact distance $L_p(q, v) = \sqrt[p]{|q.x - v.x|^p + |q.y - v.y|^p}$, in some cases we can give the answer with less computation. Denote $|q.x - v.x|$ and $|q.y - v.y|$ shortly by δ_x and δ_y, respectively, obviously $0 \leq \delta_x < 1$. So there exists a $t_x \in \{0, 1, \ldots, 9\}$ such that

$\frac{t_x}{10} \leq \delta_x < \frac{t_x+1}{10}$ and thus $c[t_x] \leq \delta_x^p < c[t_x + 1]$. For the same reason, there exists a t_y such that $c[t_y] \leq \delta_y^p < c[t_y + 1]$. Consequently,

$[\ l_b \equiv c[t_x] + c[t_y] \leq L_p^p(q, v) = \delta_x^p + \delta_y^p < c[t_x + 1] + c[t_y + 1] \equiv u_b\ ]$

This tells us that l_b and u_b are a lower bound and an upper bound of the distance between q and v, respectively. In other words, if $l_b > r^p$ then we know that the real distance will never be less than r^p; hence the vector can be safely discarded. On the other hand, if $u_b \leq r^p$ is satisfied then v is added to the answer set immediately. Noting that both the upper bound and the lower bound can be simply assembled by looking up the pre-computed array $c[\cdot]$, most vectors are judged without the expensive p-power calculation.

Further, back to Figure 5(b), the $(u_b,\ l_b)$ of $v_4,\ v_5$ and v_6 are $(c[1],\ c[1]+c[2])$, $(0,\ c[1]+c[1])$ and $(c[2],\ c[1] + c[3])$, respectively. Vectors locate to the left of, or under, q does not mean negative coordinates in this virtual grid net. In other words, when we analyse the distance, we can imagine such vectors in their symmetric side and consider only the positive sub-plane.

The image of the range query and the k Nearest Neighbors (k-NN) query processing is illustrated in Figure 6. In a), v_1 is included in the answer set automatically because even its upper bound to q ($L_p^p(q, p1) = c[3] + c[1]$) is less than r^p. On the other hand, v_2 is discarded because its lower bound ($L_p^p(q, p2) = c[4] + c[3]$) has already exceeded r^p. Unfortunately, whether v_3 satisfies the range condition is unknown and the real distance must be examined.

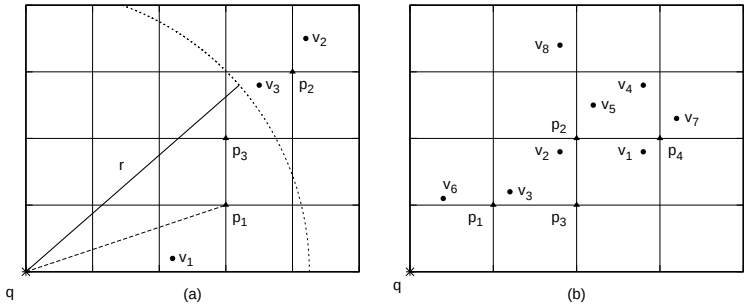


Fig. 6 (a) Examples of Range query, v_1 is answer while v_2 is not. v_3 needs a further check. (b) Example of 2-NN query

Nevertheless, a more detailed partition of [0, 1) to larger $c[\cdot]$ can reduce such a v_3 area easily. Because $c[\cdot]$ is not stored in the secondary storage, the total extra cost to double the partition is double computation of $c[\cdot]$ and the in-memory array of it. The VA-file, though good for some k-NN, can never be good for the range query.

In b), suppose that database V consists of eight vectors $v_1, v_2 \ldots, v_8$ being accessed in the order of their subscriptions and the 2-NN of the given q from V is required. This k-NN query is processed in two phases. The first phase excludes those vectors which have no possibility to be k-NN. In filtering only the bounds of the distance between each vector and q are necessary. The second phase refines the candidates of the previous phase and gives the answer to the k-NN.

To process 2-NN, firstly, the lower bounds of v_1, v_2 ($l_b(v_1)$ and $l_b(v_2)$) are obtained by looking up $c[\cdot]$ as above. Meanwhile, v_1, v_2 are taken as the first two candidates. Their upper bounds and lower bounds are sorted in ascending order, respectively. Now that there are k (here, 2) upper bounds found, ideally the newly scanned approximation can be discarded or can replace existing ones.

When v_3 is encountered, its lower bound $l_b(v_3)(\equiv\ L_p^p(q, p1))$ is compared with the existing larger (hence v_1's) upper bound $u_b(v_1)(\equiv\ L_p^p(q, p4))$. Because the former one is smaller, v_3 is added to the candidates because we are not sure currently which of v_3 and v_1 has a smaller real distance to q. On the other hand, the next encountered v_4 is discarded because $l_b(v_4) \geq max\{u_b(v_2), u_b(v_3)\}$. Therefore the real distance $L_p(q, v_4)$ between v_4 and q can never be less than that of either v_2 or v_3. Since there already exist at least two other nearer vectors to q than v_4, v_4 will never have chance to be 2-NN of q.

Similarly, $v_5,\ v_7$ and v_8 are discarded. On the other hand, v_6 remains because its lower bound is less than that of v_5

So after the filtering phase, v_1, v_2, v_3, v_6 are left as candidates. Then in the refinement phase, the real distance between v_i and q is examined for $i = 1, 2, 3, 6$. This examination finally decides that v_3 and v_6 are the answers to 2-NN of q.

4 Indexing the Expensive Function

In Table 1 we first give the notations that will be used throughout this paper, though some have been used in the previous section.

Let V and q be as in Table 1, r be a real number and k be a natural number. A range query is to find those vectors in V within distance r from q, with respect to L_p. A k-NN query aims at finding the k nearest vectors in V to q with respect to L_p.

A range query is formally expressed by finding the answer set V_r where

$V_r = \{v | v \in V, L_p(q, v) \leq r\}$

On the other hand, the answer set V_k of a k-NN query satisfies all the following conditions all together. The first condition says that there are at least k vectors in the answer set. By the second condition, V_k becomes less than k vectors if the farthest vector(s) are removed. This happens when several vectors have exactly same distance from q.

$\forall v \in V_k, \forall u \in (V - V_k), L_p(q, v) < L_p(q, u)$ and

$|V_k| \geq k \wedge |V_k - arg\ max_{v \in V_k}\{L_p(q, v)\}| < k$

Since the exponent is computationally expensive, it is meaningless to compute each element δ_d^p of the arbitrary norm L_p by its original definition. We generalize the description of the processing mentioned in Section 3 and develop the following efficient solution.

The range query algorithm

First of all, the Function *get-bound* is commonly used in both the range query algorithm and the k-NN algorithm, so we isolate it for readability. The action of this function is clear and needs no explanation.

Function get-bound(v, q, $c[B]$)
$v = (v.x_1, \ldots, v.x_D)$ is a vector of V,
$q = (q.x_1, \ldots, q.x_D)$ is the query vector.

begin
  for $d = 1, 2, \ldots, D$ **do**
    $t_d \leftarrow \lfloor |q.x_d - v.x_d| \times B \rfloor$, //quantify coordinates to *knot*s
  end for
  $l_b(v) \leftarrow \sum_{d=1}^{D} c[t_d],\ u_b(v) \leftarrow \sum_{d=1}^{D} c[t_d + 1]$
  return l_b, u_b
end

Function 1: Computation of the Upper/Lower Bounds of a vector to query q.

Our algorithm for processing range queries is very simple. As in Algorithm Range-query (Alg. 1), it scans each vector, assembling its bounds by calling Function *get-bound*, then comparing them with the given range r. Naturally, instead of comparing $L_p(q, v)$ with r for all v of V, it is much cheaper to compare $L_p^p(q, v)$ with r^p because $L_p^p(q, v)$ is assembled from $c[\cdot]$ straightforwardly. This is reflected with line 2 in the algorithm. Line 5 decides those vectors which are surely in the answer while line 7 excludes those vectors which never have the possibility to be the answer. If neither of the above happened, then we have to examine the real distance in line 9.

The Range-query algorithm is simple, while the difficulty comes from the estimation of an appropriate value of the parameter B. Nevertheless, by cost analysis (section 4.1), we will have a hint in given B for efficient filtering. Further, since $c[\cdot]$ is the only array we calculate on the fly and keep in the memory, doubling the split ($B \rightarrow 2B$) doubles $c[\cdot]$ which increases the overhead very slightly. On the other hand, methods which need secondary storage are not

Algorithm Range-query
Input: V, q, B as in Table 1, and r.
Output: Answer set V_r

begin
1: $c[t] \leftarrow (t/B)^p$ for $t = 1, 2, \ldots, B-1$.
2: $r \leftarrow r^p$; $V_r \leftarrow \emptyset$; //u_b, l_s are in fact bounds of r^p
3: **foreach** $v \in V$
4: $(l_b(v), u_b(v)) \leftarrow$ get-bound(v, q, c)
5: **if** $u_b(v) < r$ **then**
6: $V_r \leftarrow V_r \cup \{v\}$ //lower bound under r: be a answer.
7: **else if** $l_b(v) > r$ **then**
8: discard v; //upper bound exceeds r: cannot be answer.
9: **else if** $L_p^p(q, v) < r$ **then** // the real distance
10: $V_r \leftarrow V_r \cup \{v\}$
11: **end if**
12: **end foreach**
end

Algorithm 1: Range Query algorithm

designed for such dynamical change. An example is that the VA-file needs to re-calculate the approximation of all vectors and store the enlarged index file.

The k-NN query algorithm

The algorithm for processing k-NN queries is in the filtering phase and refinement phase. The essential difference compared with the VA-file is that we do not aim at the reduction of the I/O cost but the computation cost. Since our algorithm does not depend on any certain index, we can also process a query flexibly by changing B dynamically. We introduce heap structures in both phases. For readability, we assume that the heaps are sorted in ascending order.

The filtering phase is based on the idea that $\forall v_1, v_2 \in V, l_b(v_1) \geq u_b(v_2) \Rightarrow L_p(q, v_1) \geq L_p(q, v_2)$. Generally, if we have k candidates in hand and the largest upper bound among them is $u_b(v_k)$, then a new encountered vector v can be safely discarded the moment we find $l_b(v) \geq u_b(v_k)$. The Algorithm kNN-filtering (Alg. 2) describes this processing formally. Before all else, $c[t]$ is obtained by $B-1$ p-power calculations. Then the first k encountered vectors along with their lower bounds sorted in ascending order are added to candidate heap $Cand$. The first k upper bounds are also inserted to heap H_u(of fix size k, and sorted). This guarantees that there are at least k candidates (line 3-7). Then each later coming vector is compared to the largest upper bound found so far ($H_u[k]$ and line 10). The vector is added to $Cand$ only if its lower bound does not exceed $H_u[k]$. Adding a vector to $Cand$ also causes the replacement of H_u (line 12), which *tightens* the upper bounds contiguously.

In the algorithms, we hide some details to make the description simple and more readable. In real world, the heap H_u is implemented as fix size k to store the k upper bounds. Then for example in line 12, before an insertion to H_u is really carried out, the kth value is removed unconditionally. Overwriting the kth position without checking whether it is empty makes the execution more efficient. Moreover, it is not actually necessary that H_u is sorted. Being a heap, the largest element found in the root is enough for H_u.

Noting that when there comes a new vector having the upper bound exists in H_u, it is added to the candidate set while H_u remains unchanged. Therefore, usually several times

Algorithm kNN-filtering
Input: Vector set V and query vector q, B and k.
Output: Candidate set and bounds $Cand$

begin
1: create heaps $Cand$ and H_u
2: $c[t] \leftarrow (t/B)^p$ for $t = 1, 2, \ldots, B-1$.
3: **foreach** $v \in \{v_1, v_2, \ldots, v_k\}$
4: $(l_b(v), u_b(v)) \leftarrow$ get-bound(v, q, c)
5: insert $(v, l_b(v))$ into $Cand$ // $l_b(v)$ as sorting key
6: insert $u_b(v)$ into H_u // H_u is sorted
7: **end foreach**
8: **foreach** $v \in V - \{v_1, v_2, \ldots, v_k\}$
9: $(l_b(v), u_b(v)) \leftarrow$ get-bound$(v, q, c[B])$
10: **if** $l_b(v) \leq H_u[k]$ **then**
11: insert $(v, l_b(v))$ into $Cand$
12: insert $u_b(v)$ into H_u
13: **end if**
14: **end foreach**
15: **return** $Cand$
end

Algorithm 2: The filtering phase for k-NN Query

Algorithm kNN-refinement
Input: V, q, B, k.
Output: Answer set V_k (in heap structure).

begin
1: $Cand \leftarrow$ **Call** kNN-filtering
2: $V_k \leftarrow \emptyset$
3: **for** $i = 1, 2, \ldots, k$
4: $(v, l_b(v)) \leftarrow Cand[i]$
5: compute $L_p^p(q, v)$ //the real distance
6: insert $(v, L_p^p(q, v))$ into V_k
7: **end for**
8: **for** $(i = k+1; i \leq$ size-of$(Cand); i$++)
9: $(v, l_b(v)) \leftarrow Cand[i]$
10: **if** $l_b(v) > L_p^p(q, v_k)$ of $V_k[k]$ **then break**
11: **else** insert $(v, L_p^p(q, v))$ into V_k
12: **end for**
13: **return** $V_k[i], i = 1, 2 \ldots, k$
end

Algorithm 3: The refinement phase for the k-NN Query

more vectors than $|V_k|$ remain after the filtering phase, so it is necessary to examine the real distance. Algorithm kNN-refinement (Alg. 3) provides a sophisticated method to do this. For a similar reason as in the previous phase, to guarantee k vectors, the first k candidates in $Cand$ is added to answer set V_k unconditionally (line 3-7). Noting again that V_k is sorted by $L_p^p(q, v_i)$, instead of simply examining all the remaining vectors in $Cand$ and updating V_k, a technique is developed to accelerate the termination. Using the fact that real distance $(= u_b = u_l)$ is the tightest bound, the condition in line 10 terminates the algorithm any time a lower bound $l_b(v_i)$ $(i > k)$ is found that exceeds the k'th distance $L_p^p(q, v_k)$ of $V_k[k]$. Because $Cand$ was sorted in ascending order of l_b (in the filtering phase), vectors (say, v_j)

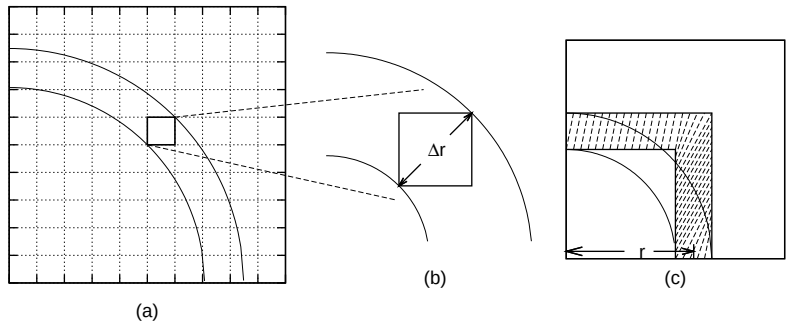


Fig. 7 Estimating the candidate volume: (a) the candidate volume *shell*, (b) the *thickness* of the shell, and (c) and upper bound of the volume of the shell.

after v_i have larger lower bounds than $l_b(v_i)$; hence their real distance to q can never be under $L_p(q, v_k)$. It is clear that such v_j can be safely discarded by expressing the relationship formally as follows.

$$L_p^p(q, v_j) > l_b(v_j) \geq l_b(v_i) > L_p^p(q, v_k)$$

4.1 Efficiency Analysis

There are various factors that influence the performance. First of all the data set. Either for the data index or for our *function index*, the efficiency of the index-based query processing depends on the dimensionality of the data space, the number of points in the database, the data distribution, and even on the correlation of dimensions. Methods other than our *function index* are especially affected by the complexity of the functions.

Our algorithms reduce the expensive function computation such as p-power by replacing it with the much cheaper multiplication and summation for those vectors judged by the bounds. In other words, the efficiency of the algorithm appears in the filtering effect; the more vectors filtered out by bounds, the more efficient it is. We analyze the filtering effect in both phases. The yardstick used here is one computational cost of the function (*eg.*$(\cdot)^p$), denoted by C_p.

The filtering effect strongly correlates with the tightness of the bounds, that is $(u_b - l_b)$. By Function get-bound (Fun. 1),

$$\Delta r = u_b - l_b = \sum_{d=1}^{D} (c[t_d + 1] - c[t_d]) = \sum_{d=1}^{D} ((\frac{t_d + 1}{B})^p - (\frac{t_d}{B})^p)$$

The maximum of this difference is $(\sum_{i=1}^{D} (\frac{t_d+1}{B} - \frac{t_d}{B})^p)^{1/p} = D^{1/p}/B$, which happens as in Figure 7 (b). In each dimension, $(u_b - l_b)$ is projected to a band of width $1/B$, and candidate vectors falling in such bands cannot be judged at the point of time. Such candidates are kept for further examination in the refinement phase in which real distance comparison is performed. The number of such candidates is in proportion to the areas (columns) of the bands (sphere shells) as shown in Figure 7 (a). Apparently, our purpose of reducing the candidate vectors to as few as possible implies the need to reduce the above area (volume). We call this the *candidate volume* and estimate it in the following way.

Range queries

In the best case when a vector is close to q, $\delta_d = |q.x_d - v.x_d|$ is close to the *virtual* original point and the candidate volume is 1. In the general case, the volume is given by the "shell" of a hyper sphere

$Vol(u_b, r) - Vol(l_b, r)$

where $Vol(D, r)$ is given by using the expression of the volume of the hyper sphere as Equation 1, where D is the dimensionality and r is the radius of the hyper sphere.

$$Vol(D,r) = \frac{\pi^{D/2} r^D}{\Gamma(1+D/2)} = \begin{cases} \frac{\pi^{D/2} r^D}{(D/2)!} & if\ D\ \geq\ 0\ is\ even, \\ \frac{\pi^{\lfloor D/2 \rfloor} 2^{\lceil D/2 \rceil} r^D}{D!!} & if\ D\ \geq\ 0\ is\ odd. \end{cases} \quad (1)$$

We simplify the estimation by building an upper bound of this volume. In Figure. 7 (c), the volume of the bounding box with the same "thickness" Δr is apparently an upper bound of the sphere inside it. Concentrating on the bounding box, imagining the 2-dimension case, the worst case arises when the vector is far away from q; hence the candidate volume, in terms of the number of hyper cubes of length $1/B$, will be the rim of the square, $B^2 - (B-1)^2$. Obviously the average estimation is $(B/2)^2 - ((B/2)-1)^2$. The analysis is extended to a general D dimension where generally the candidate volume is $(r+\Delta r/2)^D - (r-\Delta r/2)^D$

When B is large enough, this volume is estimated by

$\Delta r \cdot O(max(r^{D-1},\ \Delta r^{D-1}))$

Considering that the volume of the data space, in terms of hyper cubes length $1/B$, is B^D, the portion of the candidate volume against the whole dataset, is

$\Delta r \cdot O(max(r^{D-1}/B^D,\ \Delta r^{D-1}/B^D))$.

Because

$\Delta r \leq D^{1/p}/B$, and $r \leq (\sum_{i=1}^{D}(1-0)^p)^{1/p} = D^{1/p}$,

we have

$(r^{D-1}/B^{D-1}) \leq (D^{1/p}/B)^{D-1}$ and $\Delta r^{D-1}/B^{D-1} \leq (D^{1/p}/B)^{D-1}/B$

Finally,

$\Delta r \cdot O(max(r^{D-1}/B^D,\ \Delta r^{D-1}/B^D))$
$\leq \Delta r/B \cdot O(max(r^{D-1}/B^{D-1},\ \Delta r^{D-1}/B^{D-1}))$
$\leq D^{1/p}/B \cdot O(max((D^{1/p}/B)^{D-1},\ (D^{1/p}/B)^{D-1}/B))$
$= D^{1/p}/B \cdot O((D^{1/p}/B)^{D-1})$ (because $B > 1$)
$= O((D^{1/p}/B)^D)$

Since D is fixed, we conclude that even in the worst case, we need to check only a small portion of the whole dataset by assigning the value of B larger than $D^{1/p}$ several times. In other words, almost all the vectors can be judged in the first phase. It is also worth noting that the worst case happens only in the case when q is given as the vertexes (all $q.x_d$'s are binary 0 or 1); and all the data concentrate to the opposite angle of q, that is, all $v.x_d$'s are 1 or 0. The extreme case is when q is the original point, then all $N = |V|$ data vectors concentrate to a single point $(1, 1, \ldots, 1)$. However, in this case, the intersection between the dataset V and the candidate volume is only the few points around the vertex $(1, 1, \ldots, 1)$.

k-NN queries

In the first phase, the computation of $(\cdot)^p$ is necessary for $c[\cdot]$. Therefore the cost is $B \times C_p$. To estimate the number of the real distance computation in the second phase, we need to estimate the candidates left after the phase (size of H_l in Algorithm kNN-filtering (Alg. 2)).

Imaging from two extreme cases, it can be known that the number strongly depends on the distribution of the vectors: when at least some k vectors fall in the same block of q, then it is the number of all vectors in this block; when more than $N - k$ vectors distribute on the *surface* of $[0, 1)^D$, then the number may be as large as N.

As in the experiments for uniform distributions, this number is usually several ten times larger than k. In the second phase of the Algorithm kNN-refinement (Alg. 3) with the technique that accelerates the termination, the number of vectors that need actual computation is reduced to several times of k. Using the same policy in dimension partition, the experiment in VA-file [1] holds also here. In [1], it shows that for uniformly distributed data, for $D = 50,\ k = 10,\ N = 5 \times 10^5$, and each dimension is split by 8 bits (corresponding to $B = 2^8$ in this paper), the selectivity of the filtering phase is around 0.1%. In our experiments (eg. Fig. 18), it also shows that $B = 128$ is large enough, and the efficiency keeps unchanged for $B > 128$.

4.2 Remarks on Extensions

Till now we have explained our algorithm by applying the *indexing the function* to L_p-norm and have not considered any kind of traditional data index. We would now like to put some remarks on the extensions of our approach by several directions.

Employing other distance functions

Our method can also be applied to any other distance function if it is calculated from the coordinates. We show the example of the Mahalanobis distance, which is involved in many outlier detection methods as mentioned in [22], in the following.

The Mahalanobis distance is defined as a dissimilarity measure between two random vectors $\mathbf{x}$ and $\mathbf{y}$ following the same distribution with the covariance matrix S :

$d(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^T S^{-1} (\mathbf{x} - \mathbf{y})}$.

The Euclidean distance can be considered as a reduction of the Mahalanobis distance when S is the identity matrix. If the covariance matrix is diagonal, then the resulting distance measure is called the normalized Euclidean distance. Though the calculation of S^{-1} is time consuming, it is not necessary to calculate it every time a query is evaluated. Standardizing the coordinates to $[0,\ 1)$, then the corresponding S^{-1} can be pre-computed once and stored. With this, the values of the "knots" for each dimension can also be pre-computed and used as the bounds in the similar way. The storage cost is $|V|^2 \times$size-of-floating-point-number as well as $D \times$size-of-$c[]$. One limitation of distance functions such as the Mahalanobis distance is that they are not suitable for dynamic (eg. stream) data.

We would also like to emphasize that the more expensive the function is, the more benefit it obtains from using our method.

Combining with other Indexes

Since our approach concentrates on the cost of the function computation, it differs from other approaches based on data index. Our algorithms find their role in fields with expensive functions which attract attention in [23,24]. For well indexed data using traditional methods such as surveyed in [25], our algorithms can be applied to the data set after filtering by data indexing. It will play an active part in the new kinds of searching that need more distance calculations such as ANN([26]) and RNN([27]).

For high dimensional data vectors, the VA-file ([1]) based on the sequential scan is known to be more efficient. To such methods, our algorithms are applied in their original styles to (eg. VA-file of) V.

On non-uniform distribution data

For the original case where distances on all coordinates are computed by a single function, different distributions do not affect the performance significantly. For instance, doubling the number of grids in each dimension from 1000 to 2000 increases the calculation of the function by 1000, which is ignorable compared to the huge number of high dimensional data vectors in V. It is worth noting that doubling the number of grids makes the D dimensional space separated to $1/2^D$ thus can enhance the filtering effect significantly.

However, in the case when the distance on each dimension is calculated by different functions, the calculation as above will be increased by 1000 times D. There are two kinds of approach we can take to address such cases. One is to divide the functions into "expensive" and "not expensive" and index only the expensive ones. The other is to divide the [0, 1) of a coordinate to nearly similar to the distribution of the data on this dimension.

Applying to dimension-wise distance functions

In the case when the distance on each dimension is calculated by different functions, the calculation as above will be increased by 1000 times D. As mentioned above, dividing the functions into "expensive" and "not expensive" and indexing only the expensive ones could be a feasible policy.

5 Experimental Results

Table 2 Parameters Used in the Experiments

symbol	default value	description
N	166416	the number of vectors in data set
k	10	the number of similar vectors to be searched
D	63	dimensionality
p	0.3	the degree of power
B	128	the number of *knots*
r	0.3	the radius of range queries

In this section, to show the advantage of expensive functions, we evaluate the quality of L_p of fractional p as an example. We show the efficiency of our method, the main purpose of this paper. All experimental results presented in this section are performed on an Intel-based computer system running under Linux. CPU is Intel(R) Xeon (R) 2.83 GHz and the amount of main memory is 8.0GB. Programs are implemented in the C++ language.

5.1 A study on the Quality of L_p-norm for k-NN Search

As mentioned in the beginning, the parameter p in the L_p-norm is sensitive in high dimensional space. Using the fractional p can give better results than other distance metrics, such as Euclidean distance (p=2). Responding to this argument, we perform an extensive study to capture the behavior of the L_p-norm distance functions varying the parameter p over different datasets. We also test the quality of k-NN search applying different L_p-norm distances, and construct the precision versus recall graphs as well. This section is composed of the primary settings and the results of study.

5.1.1 Primary settings

We use seven real datasets from the UCI machine learning repository[2]. All of these datasets are prepared for classification tasks which have a large number of feature variables, and a class label to identify the classification. The information of these datasets are summarized in Table 3.

Table 3 The real datasets used in the experimental study.

Name	Features (Dimensionality)	Cardinality	Classes
Musk	166	476	92
Breast Cancer (wdbc)	30	569	2
Breast Cancer (wpbc)	33	194	2
Ionosphere	34	351	2
Wine (Red)	11	1599	6
Wine (White)	11	4898	7
Image Segmentation	19	2100	7

In order to capture the trend of changing the parameter p in L_p, and to evaluate the quality of the similarity search applying L_p to k-NN, we used the following two measures:

1. Accuracy ratio of label matching: This is the primary measure that we used to analyse the trend when different p's are used in the L_p. Designated by the providers, the class label of each object in the datasets is known. The same as the one used in [15], the measure used here is described as follows. For each dataset, each object is chosen as the target (query) object, then the k-NN search using L_p is executed on the dataset without seeing the class label. We count the total number of the k nearest neighbors that have the same class label as the target object over all the different target objects. Then the accuracy ratio of label matching is calculated by dividing the total number of objects in the same dataset. With this measure, we compare the trend of the accuracy when differing p, which presents sensitive behavior of different similarity functions.

2. Average precision versus recall: As mentioned in the literature [28], this is the well known measure to evaluate the ranked retrieval results. In our study, the answers to the k-NN search are ranked by their similarities to the query. Therefore, this measure can be employed to evaluate the quality of the similarity search. We randomly generated 100 queries from each dataset, and execute k-NN search varying the parameter p. Then the average precision across 100 queries is measured at the 11 standard recall levels of 0.0, 0.1, 0.2, ..., 1.0. For the computation of the precision, we checked the labels of the k objects in the answer set. The

[2] http://www.ics.uci.edu/~mlearn/MLRepository.html

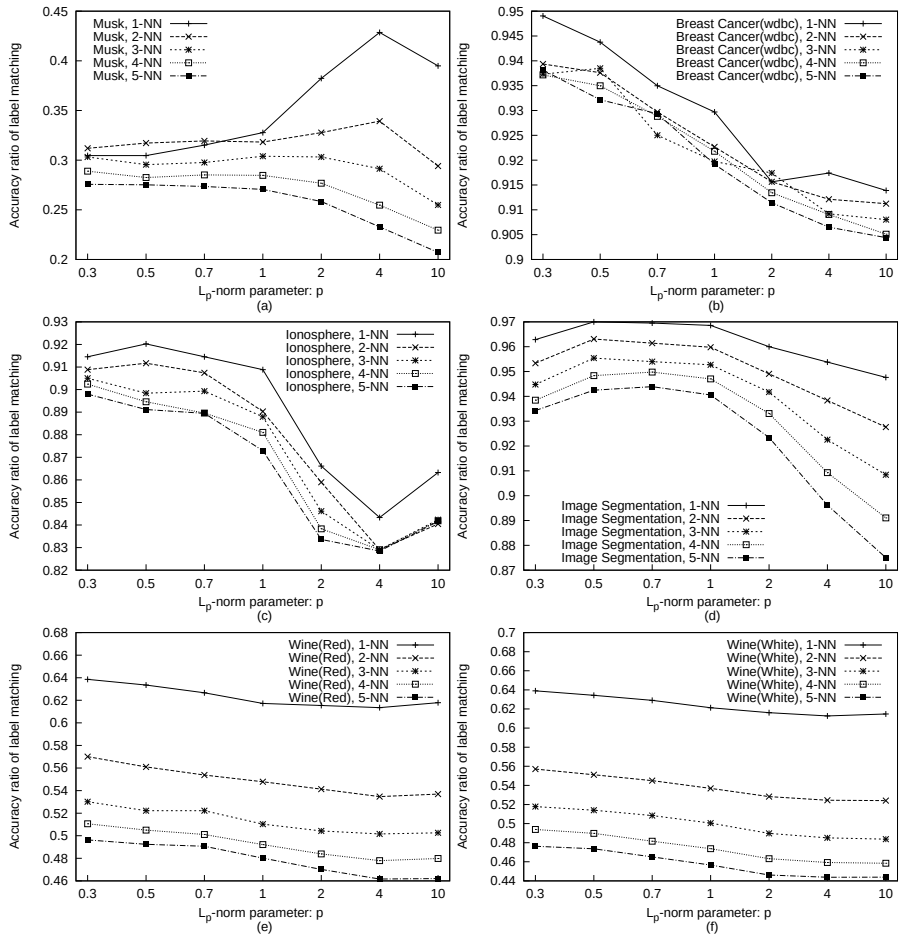


Fig. 8 Accuracy ratio of correct class label matching between the k nearest neighbors and the query target. Each graph represents the results on a dataset.

object holding the same label with the query is considered as the true positive, otherwise is false positive. To compute the recall, we assume the true answer set according the original classification in the dataset, and specified the same value of k as the number of objects in the true answer set. Finally, the continuous precision-recall curve is plotted to confirm the trends of different L_p-norm parameter p.

5.1.2 Study results

As shown in Figure 8, we executed k-NN queries on different datasets, varying the value of k from 1 to 5. Meanwhile, p are assigned values 0.3, 0.5, 0.7, 1, 2, 4, 10, in the computations, respectively. The y-axis denotes the accuracy ratio of label matching, and the x-axis denotes values of p. As illustrated by these subfigures (a)-(f), it is clear that the trends from $L_{0.3}$ to L_{10} present in the similar ways, except for a few exceptions in Figure 8(a). The major trends in these subfigures for different datasets show that the accuracy decreases when p

increases. Particularly, when p is less than 1.0, the accuracy is always better than larger p. By this observation, it is confidently that the fractional norm L_p enhances the accuracy of similarity search for these datasets.

Following the same experimental settings, we execute 100 k-NN queries on different datasets, varying the norm parameter p from the set of 0.1, 0.3, 0.5, 0.7, 1.0, 2.0, 4.0, 10.0. We illustrate the precision-recall curves in Figure 9, where each subfigure demonstrates a different dataset. The y-axis and x-axis denote the precision and recall respectively. In spite of a few exceptions, the major trends in subfigure (a)-(f) indicate that the similarity functions using fractional parameter p could achieve higher precision. Especially, the Figure 9(d) presents distinct result to support this major trend. In other words, this observation implies that preferring the fractional norm parameter p can guarantee even upgrade the quality of similarity search.

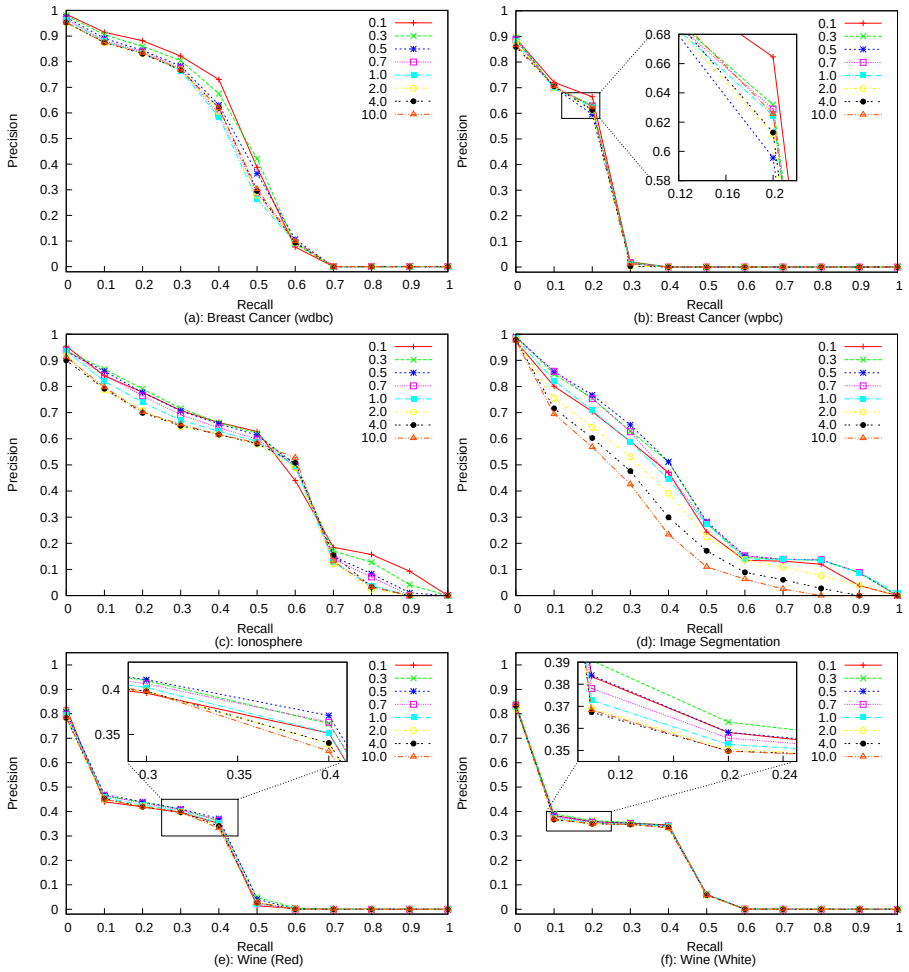


Fig. 9 Average precision vs. recall graph across 100 queries, varying different L_p-norm parameter (p) to execute k-NN query on different datasets.

5.2 A Study on Efficiency Evaluation

To confirm the efficiency of the proposed technique, we performed an experimental evaluation and compared it to the simple sequential scan and the well-known R-tree. Though many improvements such as M-tree [29], slim-tree [30] or recent NAQ-tree [31] are considered to be more efficient than R-tree, they are designed basically for metric space and are not compared here. Moreover, these methods produce non-disjoint partitions. Such indexes performs good for datasets which can be clustered hierarchically. Their performance decline when data cannot be separated well, whose extreme case is the uniformly distributed data. We believe that the comparison with one of such indexes helps to understand the properties of our method.

We used two kinds of data set for the experiments: a synthetic data set and a real data sets. The synthetic one is the set of uniformly distributed vectors. The real data set is the Corel Image Features taken from Corel GALLERY^{TM} 1,000,000 [3]. In this experiment, we extracted one set of features from all the PNG, JPG, and GIF images contained in this image collections. Totally, 166,416 feature vectors of color moments are extracted. The feature vectors of color moments are composed of 9 dimensional values as follows:

Color Moments: 9 dimensions (3×3) vectors the 9 values are: (one for each of R,G, and B in RGB color space)
- mean,
- standard deviation, and
- skewness.

The distance between the Color Moments of two images represents the dis-similarity (distance). In order to verify the effect of dimensionality, we randomly re-used some of the 9 dimensions to generate new dataset having multiples of 9 dimensions. Meanwhile, we divided the whole dataset into several sizes of sub dataset, so that we can test the experimental performance with the change of sizes. Additionally, all the data is standardized to the $[0, 1)^D$ space. We confirm that the proposed technique is consistently effective for all the data sets. The parameters used in the experiments and their default values are given in Table 2.

The extensive experiments on k-NN search compare our Function Indexing method with two related algorithms, the R-tree and the Sequential scan. In the result figures, they are shortly named as *FunIndex*, *Rtree*, and *SeqScan*, respectively. The Rtree algorithm is implemented using the Spatial Index Library [4].

We report the experimental performance by changing the following parameters, N (the size of dataset), D (the dimension of data vector), and B (the number of "knots"). The currently changing parameter is dependent on the other fixed parameters denoted in the figures. All the CPU cost is computed by issuing 100 different queries and taking their average. The queries for synthetic dataset are generated randomly holding the same distribution of the dataset, and the ones for the real dataset are randomly sampled from the original dataset.

5.2.1 Cost of exponent computation

First of all, we report the cost for computing pth powers, confirming that it is dominant when p is fractional. Figure 10 shows the time elapsed to computing d^p for fractional p. As

[3] A product of image collections published by COREL CORPORATION and COREL CORPORATION LIMITED, http://www.corel.com/

[4] http://research.att.com/~marioh/spatialindex/

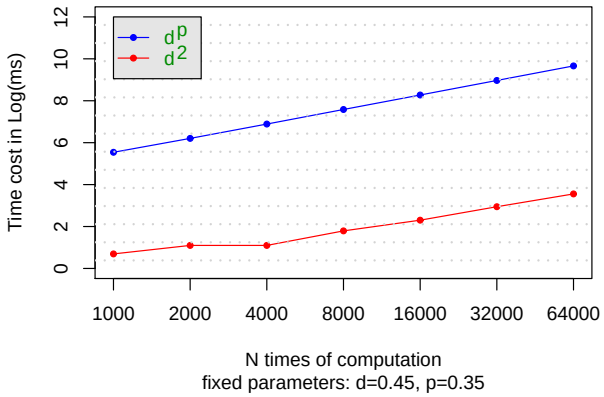


Fig. 10 Cost comparison on d^p and d^2

shown in this figure, when p is fractional, the cost is some hundreds times larger than that of Euclidean distance.

5.2.2 Range Queries: Comparison on real dataset

For range queries, we perform experiments on real dataset, while results on synthetic dataset are compared but not included here. The reason is that from the results we believe that one cannot obtain more information than what has been pointed out by so many researchers: in high dimensional space, the partitioning based index performs poor for uniform distribution datasets. Only one comparison on synthetic dataset on k-NN is shown in Figure 14.

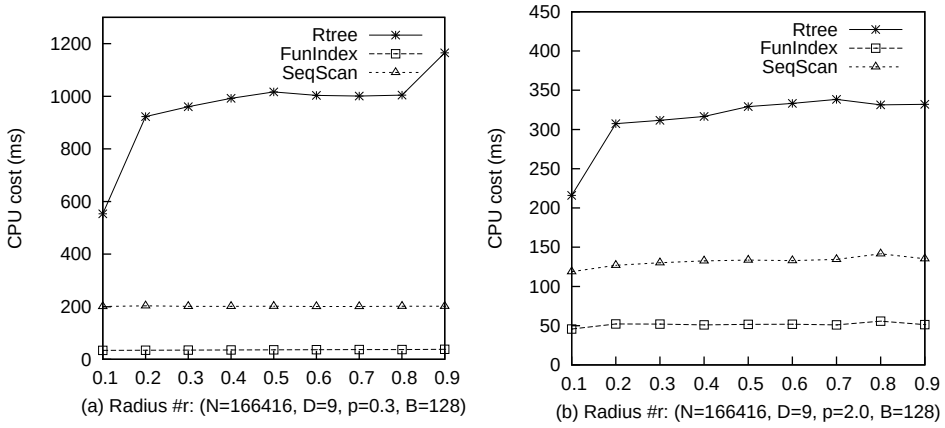


Fig. 11 Comparison on changing range r (real dataset); parameters correspond to Table 2

The experiments are illustrated in pairs in Figure 11, Figure 12, and Figure 13. In each figure, (a) and (b) give two results in correspondence with the two different remarkable p values, 0.3 and 2.0, respectively. D is fixed to 9.

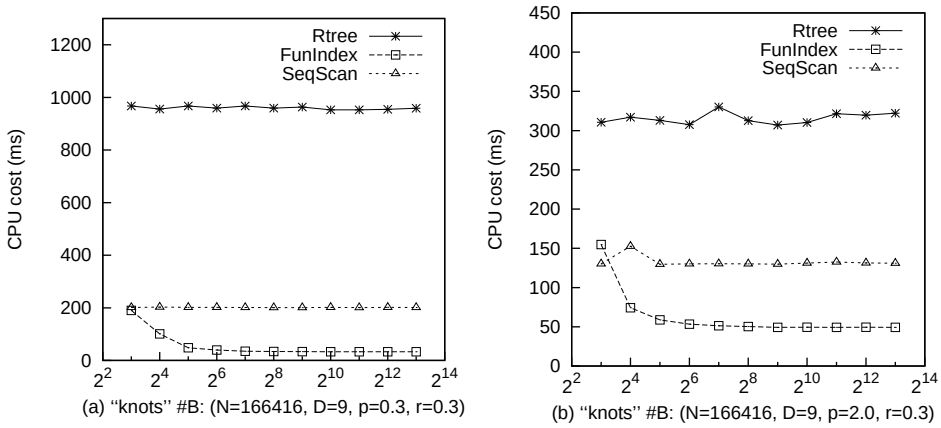


Fig. 12 Comparison on changing number of "knots" B (real dataset); parameters correspond to Table 2

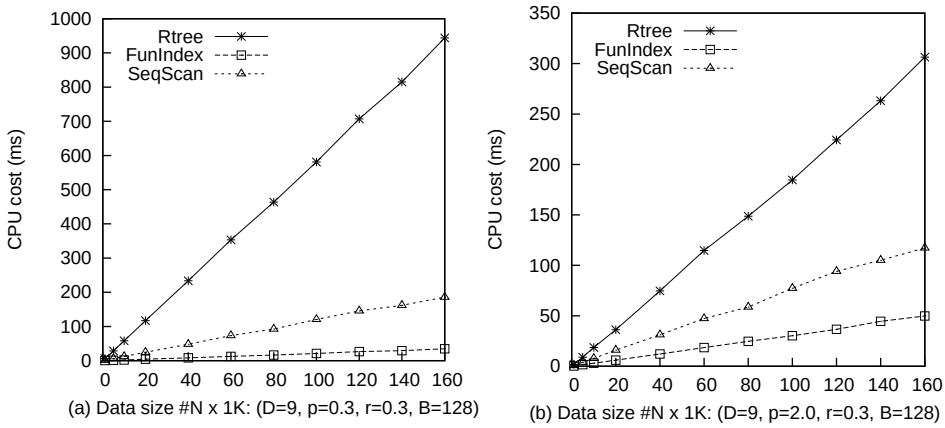


Fig. 13 Comparison on changing size of dataset N (real dataset); parameters correspond to Table 2

The first experiment is on the changing radius r, shown in Figure 11. As mentioned in cost analysis, theoretically r can be as large as $D^{(1/p)}$ when q is given as a vertex. We also test the cases of $r > 1$, and as shown in the figure, the Rtree declines when r increase. On the other hand, the sequential scan and our method perform constantly, that is, their CPU times remain unchanged when r increase. In this experiment, our method outperforms the others. So in the following experiments, we fix r to a small 0.3 from which the R-tree method performs relatively stable. The reason why we choose this r is because for extremely small r, it reduces to point query instead of range query. While for large r (eg. when $r > 0.8$ in Fig.11(a)), R-tree's performance declines.

Next, Figure 12 gives the results on the influence of parameter B, the number of *knots*. Though the cost increases very slightly when B increase in our method, the execution time convergences when B is over 128. In almost all the cases, our method outperforms the other two.

In Figure 13, we change the size of the datasets. Apparently our methods is the best and the lead is linear to N. The R-tree performs significantly worse than the sequential scan as expected.

It is worth emphasizing that all the above results show only the lower dimensional cases when D is 9, and our lead becomes more remarkable in higher dimensional experiments. Basically, the difference is in proportion to the increment of D.

5.2.3 k-NN Queries: Comparison on synthetic dataset

With each of Figure 14 to Figure 18, the experiments on k-NN are also illustrated in pairs, (a) and (b), corresponding to p of 0.3 and 2.0, respectively.

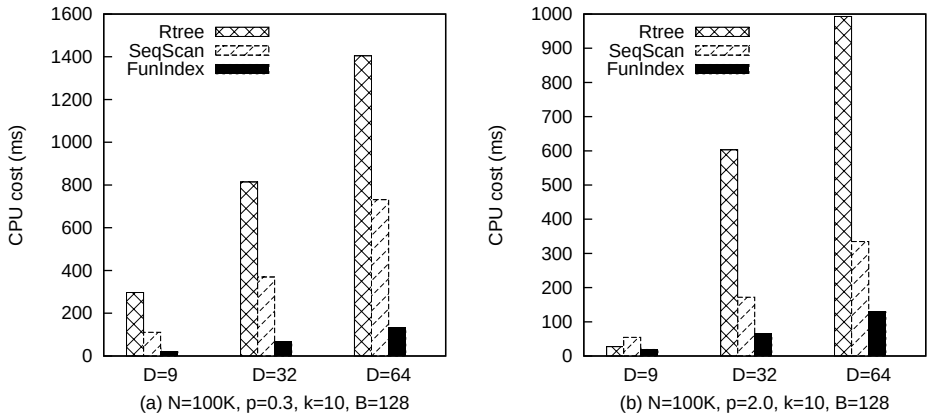


Fig. 14 Comparison on synthetic dataset conforming to uniform distribution; parameters correspond to Table 2

Figure 14 indicates the comparison results of the three algorithms executed on synthetic dataset, which is generated conforming to the uniform distribution. We fixed the size of the dataset (*ie.* N=100,000), the number of *knots* (*ie.* B=128), and issued queries by changing the parameter D. As expected, our functional indexing method is fairly successful in reducing the computation cost no matter how the dimensionality of data is changed. Comparing (a) and (b), we can see that change in the value of p to 2.0 does not diminish the superiority of our method. The computation cost of the Rtree method increases quickly with dimensionality, the case that is not the worst one happens only when $D = 9$ in L_2 space (b). On the other hand, our method always performs best with remarkable leads. The reason for these results is that the overlap of MBRs in the Rtree is rapidly increasing with the increment of dimensionality. Then to search some data, the Rtree has to search a large portion of the whole dataset. In such cases the Rtree algorithm is much worse than the sequential scan method.

5.2.4 k-NN Queries: Comparison on real dataset

In the next experiment, we evaluated the performance by varying dimensionality. The results are shown in Figure 15. With both results, we can observe that the elapsed time is linear to

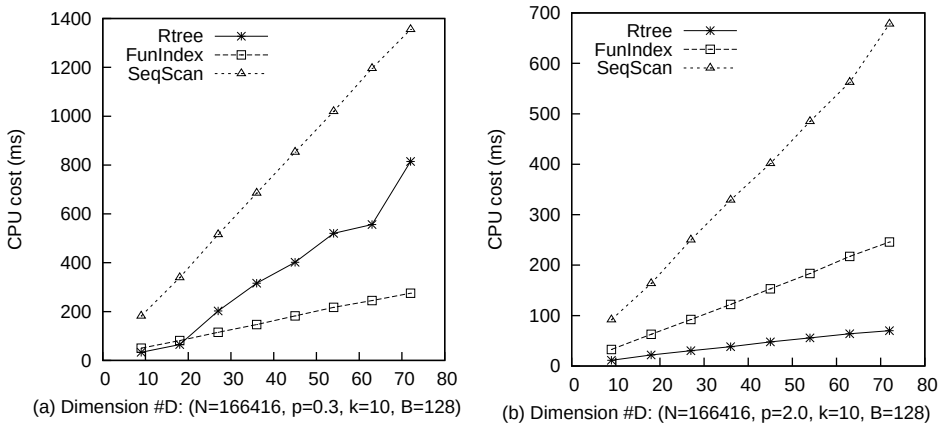


Fig. 15 Comparison on changing dimensionality D (real dataset); parameters correspond to Table 2

the dimensionality. This is because the number of computations of pth powers is linear to the dimensionality. We can also confirm that our method is stably effective. In (a) of the figure, our function indexing method is much faster than the other two algorithms and the difference enlarges with the increment of dimensionality. It is fair to explain that the Rtree declines significantly as dimensionality increases because of the expensive distance function in the $L_{0.3}$ space. However in (b), as shown in Figure 10, when distance computation is much cheaper under L_2 the R-tree is the fastest, regardless of the increasing of dimensionality. A more important reason accounting for this result is that the dataset is well-clustered. In other words, the color moments can be sensitively grouped *tightly* into different clusters. Therefore the Rtree method can divide them well with compact, disjunction MBRs. Such Rtree is constructed without high overlap, helping it to avoid the *curse of dimensionality*. Although, our method is linearly close to the Rtree method, we would like to do more experiments on different real datasets in the future. The example given in NAQ-tree [31] explains the (dis)advantage of our method and R-tree well, by showing the overlap of index blocks.

Figure 16 and Figure 17 indicate the experiment results by varying the dataset size N in low dimensional and high dimensional space. As shown in (a) of Figure 16 and 17, our method performs better than the R-tree and sequential scan methods under the expensive $L_{0.3}$ space. Although the efficiency of our function indexing method is very close to the Rtree method in the low dimensional space, the difference becomes much more significant in the high dimensional space. As expected, the well-clustered property of the corel features cannot cover the expensive distance computation in fractional p. Not surprisingly, (b) of the two figures gives a similar result to Figure 15 for the same reason to do with the corel features.

With the last experiment, we report the influence of parameter B, the number of *knots* proposed in this paper. The computation cost reduces rapidly along with the increasing of B, as shown in Figure 18. It is easy to understand that the more the numbers of "knots" are divided, the more efficiently the boundary performs for the filtering in the algorithm. Due to the same reason of the well-clustered property, in (b) where $p = 2$, the efficiency of our method increased along with B, and became stable but Rtree is still better. On the other hand,

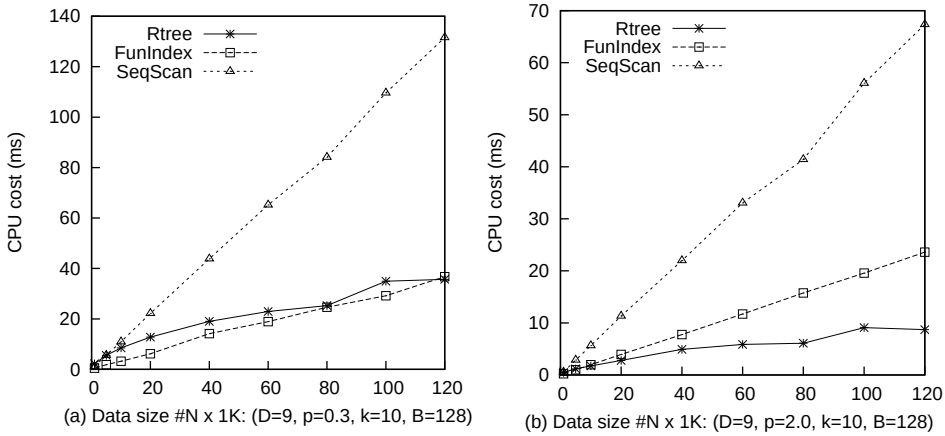


Fig. 16 Comparison on changing size of dataset N in low dimension (D=9, real dataset); parameters correspond to Table 2

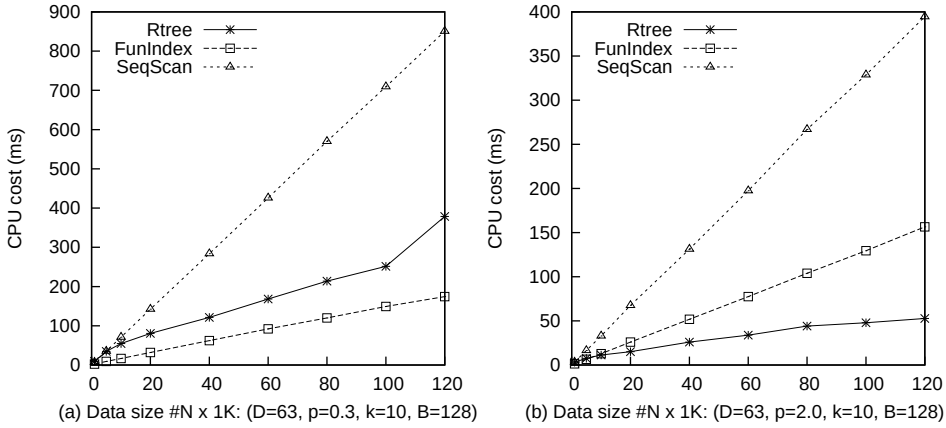


Fig. 17 Comparison on changing size of dataset N in high dimension (D=63, real dataset); parameters correspond to Table 2

in (a) where $p = 0.3$, we can see that when the value of B exceeds about 128, the efficiency of the function indexing algorithm performs slightly better than the Rtree method.

6 Conclusions

For a long time it is believed that the I/O cost dominates the performance of almost any kind of searching. Efforts are thus put on developing the data index to reduce I/O while processing searching. We figured out in this paper that the computation cost for multi-dimensional searches with expensive distance functions is also a dominating factor. To reduce such a computation cost, we developed an efficient filtering algorithm based on the new technique called *function indexing*. We also designed the range query algorithm as well as the k-NN query algorithm based on this technique. Analyses on the filtering effect of the algorithms

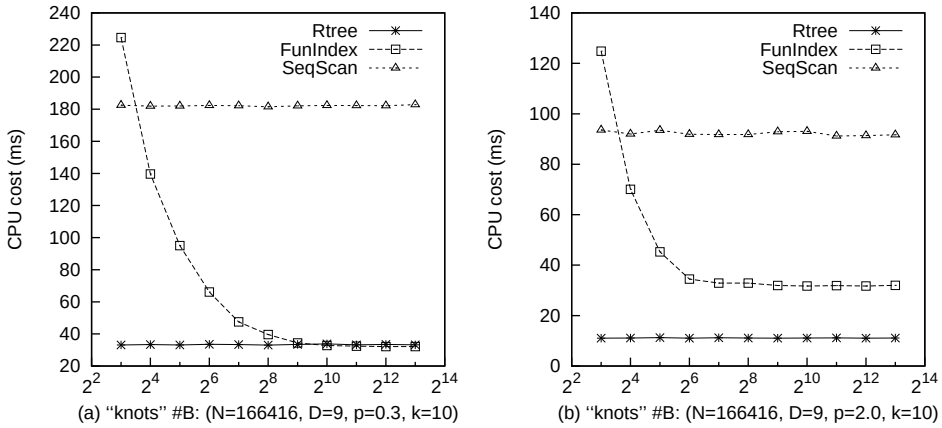


Fig. 18 Comparison on changing the number of *knots* B (real dataset); parameters correspond to Table 2

show that most of the distance computation can be avoided. As a general technique, it is also widely applicable to any kind of applications with multi-dimensional searches. Experiments on real and synthetic data confirm this efficiency.

To extend the technique, we are planning to combine it with other indexes, such as the VA-file. A more precise cost model for estimating the efficiency of our Range-query and k-NN algorithms is under construction. We recognize the necessity of the theoretical comparison between our method and the partition-based (R-tree, slim-tree, and so on) indexes. To do this, the distance between clusters as defined in [32] is an important and feasible measure. We will also investigate the effect on non-uniform data and dimension-wise distance functions.

References

1. Weber, R., Schek, H.J., Blott, S.: A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In: Proceedings of 24th International Conference on Very Large Data Bases. (1998) 194–205
2. Berchtold, S., Böhm, C., Keim, D., Kriegel, H.P.: The X-tree: An index structure for high-dimensional data. In: Proceedings of 26th International Conference on Very Large Data Bases. (1996) 28–39
3. Berchtold, S., Ertl, B., Keim, D.A., Kriegel, H.P., Seidl, T.: Fast nearest neighbor search in high-dimensional space. In: Proceedings of the 14th International Conference on Data Engineering. (1998) 209–218
4. Berchtold, S., Keim, D., Kriegel, H.P.: The pyramid-technique: Towards breaking the curse of dimensional data spaces. In: Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data. (1998) 142–153
5. Böhm, C., Berchtold, S., Keim, D.A.: Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. ACM Computing Surveys **33**(3) (2001) 322–373
6. Ferhatosmanoglu, H., Tuncel, E., Agrawal, D., Abbadi, A.E.: Vector approximation based indexing for non-uniform high dimensional data sets. In: Proceedings of the ACM International Conference on Information and Knowledge Management. (2000) 202–209
7. Berchtold, S., Böhm, C., Keim, D., Kriegel, H.P.: A cost model for nearest neighbor search in high-dimensional data space. In: ACM PODS Symposium on Principles of Database Systems. (1997) 78–86
8. An, J., Chen, H., Furuse, K., Ohbo, N.: Cva-file: An index structure for high-dimensional datasets. the Knowledge and Information Systems Journal **7**(3) (2005) 337–357
9. Chen, H., An, J., Furuse, K., Ohbo, N.: C^2VA:trim high dimensional indexes. In: Proc. WAIM2002. (2002) 303–315

10. Moise, G., Zimek, A., Kröger, P., Kriegel, H.P., Sander, J.: Subspace and projected clustering: experimental evaluation and analysis. Knowl. Inf. Syst. **21**(3) (2009) 299–326
11. Song, G., Cui, B., Zheng, B., Xie, K., Yang, D.: Accelerating sequence searching: dimensionality reduction method. Knowl. Inf. Syst. **20**(3) (2009) 301–322
12. Faloutsos, C., Lin, K.I.: Fastmap: A fast algorithm for indexing,data mining and visualization of traditional and multimedia datasets. In: Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data. (1995) 163–174
13. Jolliffe, I. In: Principal Component Analysis. Springer, New York, NY (1986)
14. McCreight, E.M.: A space-economical suffix tree construction algorithm. J. ACM **23**(2) (1976) 262–272
15. Aggarwal, C., Hinneburg, A., Keim, D.A.: On the surprising behavior of distance metrics in high dimensional spaces. In: Proceedings of the 8th Int. Conf. on Database Theory. (2001) 420–434
16. Beyer, K.S., Goldstein, J., Ramakrishnan, R., Shaft, U.: When is "nearest neighbor" meaningful. In: Proceedings of the 7th Int. Conf. on Database Theory. (1999) 217–235
17. Hinneburg, A., Agrawal, D., Keim, D.A.: What is the nearest neighbor in high dimensional spaces? In: Proceedings of the 26th VLDB Conference. (2000) 506–515
18. Yi, B., Faloutsos, C.: Fast time sequence indexing for arbitrary L_p norms. In: Proceedings of 26th International Conference on Very Large Data Bases. (2000) 385–394
19. Skopal, T., Bustos, B.: On index-free similarity search in metric spaces. In: DEXA. (2009) 516–531
20. Zhang, Z., Ooi, B.C., Parthasarathy, S., Tung, A.K.H.: Similarity search on bregman divergence: Towards non-metric indexing. PVLDB **2**(1) (2009) 13–24
21. Chen, H., Liu, J., Furuse, K., Yu, J.X., Ohbo, N.: Indexing the function: An efficient algorithm for multi-dimensional search with expensive distance functions. In: ADMA. (2009) 67–78
22. Fan, H., Zaïane, O.R., Foss, A., Wu, J.: Resolution-based outlier factor: detecting the top- most outlying data points in engineering data. Knowl. Inf. Syst. **19**(1) (2009) 31–51
23. Chen, H., Yu, X., Yamaguchi, K., Kitagawa, H., Ohbo, N., Fujiwara, Y.: Decomposition – an approach for optimizing queries including adt functions. Information Processing Letters **43**(6) (1992) 327–333
24. Hellerstein, J.M.: Optimization techniques for queries with expensive methods. ACM Transactions on Database Systems (TODS) **23**(2) (1998) 113–157
25. Gaede, V., Gunther, O.: Multidimensional access methods. ACM Computing Surveys (CSUR) **30**(2) (1998) 170–231
26. Papadias, D., Tao, Y., Mouratidis, K., Hui, C.K.: Aggregate nearest neighbor queries in spatial databases. ACM Transactions on Database Systems (TODS) **30**(2) (2005) 529–576
27. Y. Tao, M.Y., Mamoulis, N.: Reverse nearest neighbor search in metric spaces. IEEE Transactions on Knowledge and Data Engineering **18**(9) (2006) 1239–1252
28. Manning, C.D., Raghavan, P., Schtze, H.: Introduction to Information Retrieval. Cambridge University Press, New York, NY, USA (2008)
29. Ciaccia, P., Patella, M., Zezula, P.: M-tree: An efficient access method for similarity search in metric spaces. In: VLDB. (1997) 426–435
30. Jr., C.T., Traina, A.J.M., Seeger, B., Faloutsos, C.: Slim-trees: High performance metric trees minimizing overlap between nodes. In: EDBT. (2000) 51–65
31. Zhang, M., Alhajj, R.: Effectiveness of naq-tree as index structure for similarity search in high-dimensional metric space. Knowl. Inf. Syst. **22**(1) (2010) 1–26
32. Pfitzner, D., Leibbrandt, R., Powers, D.M.W.: Characterization and evaluation of similarity measures for pairs of clusterings. Knowl. Inf. Syst. **19**(3) (2009) 361–394

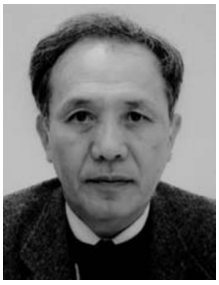
Author Biographies

Hanxiong Chen received the B.S. degree from Zhongshan University, Guangdong, China, in 1985, the M.S. degree and the Ph.D degree in computer science, from the University of Tsukuba, Japan, in 1990 and 1993, respectively. He is currently an assistant professor in the Graduate School of Systems and Information Engineering, University of Tsukuba. His research interests include data engineering, knowledge discovery, data mining and information retrieval. He is a member of ACM, IEEE-CS and IPSJ.

Jianquan Liu received the B.E. degree in computer science, from Shantou University, Guangdong, China, in 2005. From 2005 to 2006, he worked as a Development Engineer for Tencent Inc., one of the most largest Internet service portals in China. In 2009, he received the M.E. degree in computer science, from the University of Tsukuba, Japan. He is currently a Ph.D. candidate at the Department of Computer Science, Graduate School of Systems and Information Engineering, University of Tsukuba. His research interests include high-dimensional data mining, web data mining and information retrieval. He is a student member of the Database Society of Japan (DBSJ).

Kazutaka Furuse received his B.E., M.E. and Ph.D. degrees in computer science, from the University of Tsukuba, in 1988, 1990, and 1993, respectively. He was a R&D Engineer in RICOH Software Research Center from 1993 to 1996. Then he joined University of Ibaraki as a research associate until 1999. He is currently an assistant professor in the Graduate School of Systems and Information Engineering, University of Tsukuba from 2000. His research interests include materialized XML view maintenance, query processing, indexing mechanisms and architecture of advanced database systems. He is a member of ACM, IEEE-CS, IEICE and IPSJ.

Jeffrey Xu Yu received his B.E., M.E. and Ph.D. in computer science, from the University of Tsukuba, Japan, in 1985, 1987 and 1990, respectively. Dr. Yu held teaching positions in the Institute of Information Sciences and Electronics, University of Tsukuba, Japan, and the Department of Computer Science, The Australian National University. Currently, he is a Professor in the Department of Systems Engineering and Engineering Management, the Chinese University of Hong Kong. Dr. Yu served as an associate editor of IEEE Transactions on Knowledge and Data Engineering, and is serving as a VLDB Journal editorial board member, and an ACM SIGMOD Information Director. His current main research interest includes graph database, graph mining, keyword search in relational databases, XML database, and Web-technology. He has published over 200 papers including papers published in reputed journals and major international conferences.



Nobuo Ohbo received the Dr.Sc. degree from the University of Tokyo in 1979. He joined the Institute of Information Sciences and Electronics, University of Tsukuba, in 1980, where he was a professor heading the database research group. Before joining the University of Tsukuba, he was a research associate in the University of Tokyo from 1970 to 1980. Currently he is an emeritus professor in the Graduate School of Systems and Information Engineering from 2009. His current research interests include multimedia database systems, data mining and information retrieval. He is a member of ACM, IEEE-CS and IPSJ.