Decreasing Radius K-Nearest Neighbor Search using Mapping-based Indexing Schemes

by

Qingxiu Shi and Bradford G. Nickerson

TR06-179, June 1, 2006

Faculty of Computer Science
University of New Brunswick
Fredericton, N.B. E3B 5A3
Canada

Phone: (506) 453-4566
Fax: (506) 453-3566
Email: fcs@unb.ca
www: http://www.cs.unb.ca

Abstract

A decreasing radius k-nearest neighbor search algorithm for the mapping-based indexing schemes is presented. We implement the algorithm in the Pyramid technique and the iMinMax(θ), respectively. The Pyramid technique divides d-dimensional data space into $2d$ pyramids, and the iMinMax(θ) divides the data points into d partitions. Given a query point q, we initialize the radius of a range query to be the furthest distance of the k candidate nearest neighbors from q in the pyramid (partition) which q is in, then examine the rest of the pyramids (partitions) one by one. After one pyramid (partition) is checked, the radius of the range query decreases or remains the same. We compare the decreasing radius k-nearest neighbor search algorithm with the increasing radius k-nearest neighbor search algorithm using the Pyramid technique and the iMinMax(θ). Moreover, the decreasing radius k-nearest neighbor search performance of these two mapping-based indexing schemes is compared to the BBD-tree, the R^*-tree and naive search. Experimental results show that our decreasing radius k-nearest neighbor search algorithm for the mapping-based indexing schemes is efficient when $d \leq \log_2 n$.

Contents

Abstract **ii**

1 Introduction **1**
1.1 Related Work 1
1.2 Our Results 4

2 Two Mapping-based Indexing Schemes **5**
2.1 The Pyramid Technique 5
2.1.1 Mapping d-d Data into 1-d Value 5
2.1.2 Mapping Range Queries 6
2.2 iMinMax(θ) 8

3 DR KNN Search Algorithms **12**
3.1 DR Pyramid KNN Search Algorithm 12
3.2 DR iMinMax KNN Search Algorithm 14

4 Experimental Evaluation **17**
4.1 DR vs. IR 17
4.2 DR Pyramid Technique vs. Other Techniques 19

5 Conclusions **21**

References **21**

List of Tables

List of Figures

1 (a) A set of points in 2-d data space $[0, 1]^2$ (the numbers in the triangles are the pyramid numbers), and (b) the corresponding B^+-tree (the maximum number of keys is 4, and the point insertion order is (0.2,0.7), (0.1,0.3), (0.3,0.4), (0.2,0.1), (0.4,0.2), (0.5,0.3), (0.6,0.3), (0.8,0.4), (0.7,0.6), (0.9,0.7), (0.7,0.8), (0.5,0.9)). 6
2 Algorithm for calculating the pyramid value pv_v of a point v, adapted from [4]. 7

3 Orthogonal range search algorithm for the Pyramid technique, adapted from [4]. $MIN(a, b)$=0, if $a \leq 0 \leq b$, else $MIN(a, b)$ = $min(|a|, |b|)$. . . . 9
4 The data space and the query rectangle W (the black area is the region of W, and the cross-hatched area is the region needed to be visited during range search in addition to W) . . 10
5 A 2-d iMin example. (a) A set of points in 2-d data space $[0, 1]^2$, which is divided by the diagonal y=x to two right-angled triangles, numbered 0 and 1 (i.e. d_{min}), respectively. (b) the corresponding B^+-tree (the maximum number of keys is 4, and the point insertion order is (0.2,0.7), (0.1,0.3), (0.3,0.4), (0.2,0.1), (0.4,0.2), (0.5,0.3), (0.6,0.3), (0.8,0.4), (0.7,0.6), (0.9,0.7), (0.7,0.8), (0.5,0.9)). . . . 10
6 Range search algorithm for iMinMax(θ), adapted from [19]. . 11
7 Sample search space in 2-d data space $[0, 1]^2$ (the black area is the region of the query rectangle W, and the cross-hatched area is the region needed to be visited during range search in addition to W) 11
8 An illustration of a 2-d KNN search using increasing radius approach. (a) shows the KNN search with an initial radius, and (b) shows the KNN search with an increasing radius. The cross-hatched region is the search region. 12
9 The decreasing radius Pyramid KNN search algorithm. 14
10 An illustration of a 2-d KNN query processing. The union of cross-hatched region in these four figures is the total search region for the KNN query. 15
11 The decreasing radius iMinMax KNN search algorithm. 16
12 An illustration of a 2-d KNN query processing. The union of cross-hatched region in (a) and (b) is the total search region for the KNN query. 16
13 KNN search in the Pyramid technique, where n=1,000,000, 2$\leq$ d $\leq$20, and k=10. 18
14 KNN search in the iMinMax(θ), where θ=0, n=1,000,000, 2$\leq$ d $\leq$20, and k=10. 18
15 n=1,000,000, 2$\leq$ d $\leq$20, and k=10. 19
16 n=1,000,000, d=16, and 5$\leq$$k$$\leq$ 50. 20
17 100,000$\leq$ n $\leq$1,000,000, d=16, and k=20. 20

1 Introduction

Similarity searching is very important for many applications, such as computer graphics, multimedia databases [10][12][24], geographic information systems, knowledge discovery and data mining [11], and computer aided design systems [17][22], etc. Similarity searching often reduces to finding the k nearest neighbors to a query point. Given a set S of n d-dimensional (d-d) data points and a query point q, the k-nearest neighbor (KNN) search is to find a subset $S' \subseteq S$ of $k \leq n$ data points such that for any data point $u \in S'$ and $v \in S - S'$, dist(u,q)$\leq$dist(v,q). Several KNN search algorithms have been discussed in [6][7].

1.1 Related Work

In [21], the authors presented a branch-and-bound search algorithm for processing KNN queries for R-trees. R-trees were proposed as a natural extension of B-trees in higher than one dimensions [13]. An internal node of the R-tree contains entries of the address of a child node and a minimum bounding rectangle (MBR) of all rectangles which are entries in that child node. A leaf node contains entries of the pointer to data object and an MBR which is the enclosing rectangle of that data object. Roussopoulos et al. defined two distance functions, MINDIST and MINMAXDIST. MINDIST is the minimum distance from the query point to the enclosed MBR. MINMAXDIST is the minimum of the maximum possible distances from the query point to the vertices of the MBR. When processing KNN search, MINDIST and MINMAXDIST are used for ordering and pruning the search. The KNN search algorithm can be applicable to other R-tree-like structures, e.g. the R^*-tree [3], the SS-tree [23], the X-tree [5] and the SR-tree [16].

The M-tree [8] was proposed to organize and search large data sets from a generic metric space, i.e., where object proximity is only defined by a distance function satisfying the positivity, symmetry, and triangle inequality postulates. The M-tree partitions objects on the basis of their relative distances, as measured by a specific distance function, and stores these objects into fixed-size nodes, which correspond to constrained regions of the metric space. Leaf nodes of an M-tree store all indexed objects, and internal nodes store the so-called routing objects. For each routing object O_r, there is an associated pointer, denoted $ptr(T(O_r))$, which references the root of the subtree, $T(O_r)$, called the covering tree of O_r. All objects in the covering tree of

O_r, are within the distance $r(O_r)$ from O_r, $r(O_r)>0$, which is called the covering radius of O_r. Finally, a routing object O_r is associated with a distance to $P(O_r)$, its parent object, that is, the routing object that references the node where the O_r entry is stored. For KNN search, they use a branch-and-bound technique, similar to the one designed for R-trees [21], which utilizes a priority queue PR, a queue of pointers to subtrees where qualifying objects can be found, and a k elements array NN to contain the result.

The vector approximate file (VA-file) [20] divides the data space into 2^b rectangular cells where b denotes a user specified number of bits (e.g. some number of bits per dimension). Instead of hierarchically organizing these cells like in grid-files or R-trees, the VA-file allocates a unique bit-string of length b for each cell, and approximates data points that fall into a cell by that bit-string. The VA-file itself is simply an array of these compact, geometric approximations. When searching for the k nearest neighbors, the entire approximation file is scanned, and upper and lower bounds on the distance to the query can easily be determined based on the rectangular cell represented by the approximation. The vast majority of vectors from the search is excluded (filtering step) based on these approximations. After the filtering step, a small set of candidates remain. These candidates are then visited in increasing order of their lower bound on the distance to the query point q, and the accurate distance to q is determined. However, not all candidates must be accessed. If a lower bound is encountered that exceeds the k-th nearest distance seen so far, the KNN search stops.

In [9], a multi-tier index structure, called Δ-tree, was proposed to speed up processing of high-dimensional KNN queries in the main memory environment. Each tier in the Δ-tree represents the data space as clusters in different number of dimensions and tiers closer to the root partition the data space using fewer number of dimensions. The leaf level contains the data at their full dimensions. The numbers of tiers and dimensions are obtained using the Principal Component Analysis [15] technique. Each level of the tree serves to prune the search space more efficiently as the reduced dimensions can better exploit the small cache line size. Moreover, the distance computation on lower dimensionality is less expensive. An extension, called Δ^+-tree, was proposed that globally clusters the data space and then further partitions clusters into small regions to reduce the search space. Two data structures are used to facilitate KNN search: a priority queue to maintain entries in non-descending order of distance. Each item in the queue is an internal node the Δ-tree. The second is the list of KNN candidates. The distance between the k-th nearest neighbor and the query point is used to prune away points that are further away.

In [14], the authors proposed an efficient method for KNN search in multi-dimensional data space, called iDistance. iDistance partitions the data and defines a reference point for each partition. The data points in each partition are transformed into a 1-d data space based on their similarity with respect to the reference point. Then the distance of each data point is indexed to the reference point of its partition. A B^+-tree is used to index this distance. For a KNN query centered at q, a range query with radius r is issued. The iDistance KNN search algorithm searches the index from the query point outwards, and for each partition that intersects with the query sphere, a range query is resulted. If the algorithm finds k elements that are closer than r from q at the end of the search, the algorithm terminates. Otherwise, it extends the search radius by Δr, and the search continues to examine the unexplored region in the partitions that intersects with the query sphere. The process is repeated till the stopping condition is satisfied.

Due to the hardness of processing exact KNN queries, Arya et al. [1] [2] turn to approximate KNN search. Given a relative error bound ϵ, a point p is a $(1+\epsilon)$-approximate jth nearest neighbor to a point q if its distance from q is a factor of at most $(1+\epsilon)$ times the distance to q's true jth nearest neighbor. An answer to the approximate k-nearest neighbors query is a sequence of k distinct data points $p_1, p_2, \cdots, p_k$, such that p_j is a $(1+\epsilon)$-approximation to the jth nearest neighbor of q, for $1 \leq j \leq k$. The approximate KNN search algorithm is based on a balanced box-decomposition (BBD) tree. Data space is recursively subdivided into a collection of cells, each of which is either a d-d rectangle or the set-theoretic difference of two rectangles, one enclosed within the other. The ratio between the longest and shortest sides of rectangles is bounded. Each node of the BBD-tree is associated with a cell, and hence it is implicitly associated with the set of data points lying within this cell. Each leaf cell is associated with a single point lying within the bounding rectangle for the cell. The leaves of the tree define a subdivision of space. The KNN search algorithm locates the leaf cell containing the query point q, and then enumerates cells in increasing order of distance from q. The k candidates are stored in a balanced binary search tree sorted by their distance to q. The search terminates as soon as the distance from the current cell to q exceeds $r_k/(1+\epsilon)$, where r_k is the distance of the k-th nearest neighbor to q.

1.2 Our Results

In Section 2 we have a closer look at the Pyramid technique [4] and the iMinMax(θ) [19], and present their orthogonal range search algorithms. To facilitate the KNN search, we make some modifications on these two mapping-based indexing schemes. The Pyramid technique divides d-d data space into $2d$ pyramids, and the iMinMax(θ) divides the data points into d partitions. We proposed decreasing radius KNN search algorithms for the Pyramid technique and the iMinMax(θ) in Section 3. The radius r of a range query is initialized to be the distance of the k-th nearest neighbor candidate after examining the data points in the pyramid (partition) the query point q in. A query square W centered at q with side length $2r$ is generated, then an orthogonal range search is performed in one of the remaining $2d$-1 pyramids (partitions). If the search finds some data points closer to q than r, the k candidate nearest neighbors answer set is updated, and r is reset to be the current furthest distance of the k candidates. The search continues to examine one of the unexplored pyramids (partitions), and the radius r decreases if at least one new candidate is found. The process is repeated until all pyramids (partitions) are examined.

In Section 4, we conduct a series of experiments to evaluate the decreasing radius KNN search performance. We first compare the increasing radius KNN search approach with the decreasing radius KNN approach using the Pyramid technique and the iMinMax(θ), then we compare the decreasing radius Pyramid and iMinMax KNN search performance with the R^*-tree, the BBD-tree and naive search. Our experiments are performed using uniformly and randomly distributed data points from the interval $[0,1]^d$. Overall, the experimental results show that the decreasing radius KNN search algorithm outperforms the increasing radius approach, and the decreasing radius Pyramid KNN search is efficient when $d \leq \log_2 n$.

Without loss of generality, the following discussions are all based on unit space $[0,1]^d$. Distances are measured using any Minkowski L_m distance metric. For any integer $m \geq 1$, the L_m-distance between points $p = (p_0, p_1, \cdots, p_{d-1})$ and $q = (q_0, q_1, \cdots, q_{d-1})$ in R^d is defined to be the mth root of $\Sigma_{i=0}^{d-1}|p_i - q_i|^m$. In the limiting case, where $m = \infty$, this is equivalent to $max_{i=0}^{d-1}|p_i - q_i|$. The L_1, L_2 and L_∞ metrics are the well-known City Block (Manhattan) distance, Euclidean distance, and Chebyshev distance (max metrics), respectively. In our experiments, we use the Euclidean distance.

2 Two Mapping-based Indexing Schemes

In recent years, several mapping-based indexing schemes have been proposed to improve the performance of range search in high-dimensional data space, e.g. the Pyramid technique [4] and the iMinMax(θ) [19]. The basic idea is to transform the d-d data points into 1-d values, and then store and access the 1-d values using a B^+-tree. A d-d range query is mapped to a union of 1-d range queries. Based on the similarity between these schemes, Zhang et al. [25] proposed a generalized structure for multidimensional data mapping and query processing. The mapping-based indexing scheme overcomes the high dimensionality problem. In the following subsections, we had a close look at the Pyramid technique and the iMinMax(θ), and made some little modifications to facilitate the KNN search.

2.1 The Pyramid Technique

The basic idea of the Pyramid technique [4] is to transform the d-d data points into 1-d values, and then store and access 1-d values using a B^+-tree. The data space is divided in two steps: firstly, the data space is split into $2d$ pyramids having the center point of data space $(0.5, 0.5, \cdots, 0.5)$ as their top and a (d-1)-d surface of the data space as their base. Secondly, each of the $2d$ pyramids is divided into several partitions, each corresponding to one data block of the B^+-tree.

2.1.1 Mapping d-d Data into 1-d Value

Assume a data point $v = (v_0, v_1, \cdots, v_{d-1})$ is in pyramid i. The height h_v of the point is defined to be the distance between v and the center in dimension i mod d, i.e. $h_v = |0.5 - v_{i\ mod\ d}| \in [0, 0.5]$. As shown in Fig.1(a), the data space $[0, 1]^2$ is divided into 4 triangles, sharing the center point $(0.5, 0.5)$ as their top and one edge as base. Each triangle is assigned a number between 0 and 3. The pyramid value pv_v of v is defined as the sum of its pyramid number i and its height h_v: $pv_v = i + h_v$. The algorithm for calculating pv_v is given in Fig.2. The pyramid i covers an interval of $[i, i + 0.5]$ pyramid values and the sets of pyramid values covered by any two different pyramids are disjoint. After determining the pyramid value of v, we insert v into a B^+-tree using pv_v as a key, and store v in the corresponding leaf node of the B^+-tree (See Fig.1(b)).

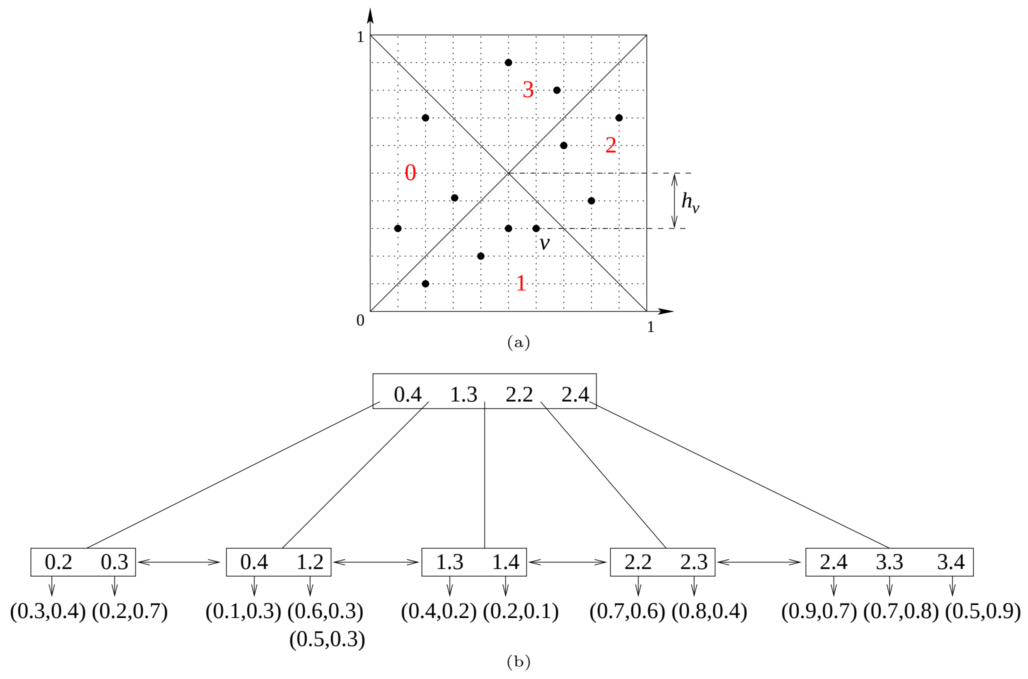


Figure 1: (a) A set of points in 2-d data space $[0,1]^2$ (the numbers in the triangles are the pyramid numbers), and (b) the corresponding B^+-tree (the maximum number of keys is 4, and the point insertion order is (0.2,0.7), (0.1,0.3), (0.3,0.4), (0.2,0.1), (0.4,0.2), (0.5,0.3), (0.6,0.3), (0.8,0.4), (0.7,0.6), (0.9,0.7), (0.7,0.8), (0.5,0.9)).

The internal nodes of a B^+-tree of order M contain between M and $2M$ keys. An internal node of the B^+-tree with m keys has $m+1$ child pointers. The leaf node with m keys has one left pointer (added for the purposes of KNN search and practical efficiency), one right pointer and m data point pointers. The left (right) pointer points to the immediate left (right) sibling node at the leaf level in the B^+-tree.

2.1.2 Mapping Range Queries

Given a query rectangle $W = [L_0, H_0] \times [L_1, H_1] \times \cdots \times [L_{d-1}, H_{d-1}]$, the key $v = (v_0, v_1, \cdots, v_{d-1})$ is in the range iff $v_i \in [L_i, H_i]$, $\forall i \in (0, 1, \cdots, d-1)$. We define $\overline{W} = [\overline{L}_0, \overline{H}_0] \times [\overline{L}_1, \overline{H}_1] \times \cdots \times [\overline{L}_{d-1}, \overline{H}_{d-1}]$, where $\overline{L}_i = L_i - 0.5$ and $\overline{H}_i = H_i - 0.5$. A pyramid p_i is intersected by W if and only if

```
PyramidValue(Point v)
 1  $d_{max} \leftarrow 0$
 2  $h_v \leftarrow |0.5 - v_0|$
 3  for $(j = 1; j < d; j \leftarrow j + 1)$
 4  do if $(h_v < |0.5 - v_j|)$
 5        then $d_{max} \leftarrow j$
 6             $h_v \leftarrow |0.5 - v_j|$
 7  if $(v_{d_{max}} < 0.5)$
 8     then $i \leftarrow d_{max}$
 9     else $i \leftarrow d + d_{max}$
10  $pv_v \leftarrow i + h_v$
11  return $pv_v$
```

Figure 2: Algorithm for calculating the pyramid value pv_v of a point v, adapted from [4].

1. $\overline{L}_i \leq -MIN(\overline{L}_j, \overline{H}_j)$, if $i < d$, and
2. $\overline{H}_{i-d} \geq MIN(\overline{L}_j, \overline{H}_j)$, if $d \leq i < 2d$

$\forall j, 0 \leq j < d$, where $MIN(\overline{L}_j, \overline{H}_j) = 0$, if $\overline{L}_j \leq 0 \leq \overline{H}_j$, else $MIN(\overline{L}_j, \overline{H}_j) = min(|\overline{L}_j|, |\overline{H}_j|)$ [4].

Then we need to calculate the interval $[h^i_{low}, h^i_{high}]$ that the pyramid values of all data points inside the intersection of W and pyramid p_i are in the interval $[i + h^i_{low}, i + h^i_{high}]$. We define a more restricted value of h_{low} than the original one in [4].

The modified query rectangle $\widetilde{W}$ for pyramid p_i $\widetilde{W}_i = [\widetilde{L}_0, \widetilde{H}_0] \times [\widetilde{L}_1, \widetilde{H}_1] \times \cdots \times [\widetilde{L}_{d-1}, \widetilde{H}_{d-1}]$, where

1. $\widetilde{L}_j = \overline{L}_j$, $\widetilde{H}_j = min(\overline{H}_j, 0)$, if $i < d$ and $j = i \mod d$, or
2. $\widetilde{L}_j = max(\overline{L}_j, 0)$, $\widetilde{H}_j = \overline{H}_j$, if $d \leq i < 2d$ and $j = i \mod d$
3. $\widetilde{L}_j = \overline{L}_j$ and $\widetilde{H}_j = \overline{H}_j$ for $0 \leq j < d$, $j \neq i \bmod d$

Given a query rectangle $\overline{W}$ and an affected pyramid p_i, the intersection interval $[h^i_{low}, h^i_{high}]$ is define as follows:

1. $h^i_{low} = 0$, if $\overline{L}_j \leq 0 \leq \overline{H}_j$, $\forall j \in \{0, 1, \cdots, d-1\}$, or

2. h^i_{low}=$min^{d-1}_{j=0} max(MIN(\widetilde{L}_{i \bmod d}, \widetilde{H}_{i \bmod d}), MIN(\widetilde{L}_j, \widetilde{H}_j))$

3. $h^i_{high} = max(|\widetilde{L}_{i \bmod d}|, |\widetilde{H}_{i \bmod d}|)$

Range search begins from the root T of the B^+-tree. If pyramid p_i ($0 \leq i \leq 2d-1$) is intersected by W, we do 1-d range search on the B^+-tree using interval $[i + h^i_{low}, i + h^i_{high}]$. When we reach the leaf level of the B^+-tree, we determine if the data points pointed by the leaf node intersect W. The detailed orthogonal range search algorithm is given in Fig.3. Fig.4 shows the region visited when an orthogonal range search is performed in the Pyramid technique.

2.2 iMinMax(θ)

For a data point $v = (v_0, v_1, \cdots, v_{d-1})$, let v_{max}=$max^{d-1}_{i=0} v_i$, and v_{min}=$min^{d-1}_{i=0} v_i$ be the maximum value and minimum value among the d dimensions of v, and let d_{max} and d_{min} denote the dimensions at which v_{max} and v_{min} occur. iMinMax(θ) [19] uses either v_{max} or v_{min} as the representative index key for the point v, and store and access the key value in a B^+-tree. The data point v is mapped to a 1-d value θ_v as follows:

$$\theta_v = \begin{cases} d_{min} + v_{min}, & \text{if } v_{min} + \theta < 1 - v_{max} \\ d_{max} + v_{max}, & \text{otherwise} \end{cases}$$

where θ is a real number. For simplicity, we call this 1-d value as iMinMax value, and we number the partition which the point v is in by $\lfloor \theta_v \rfloor$, called partition number, which is between 0 and d-1. There are d partitions altogether. There are two extremes: when $\theta \geq 1$, the transformation maps all points to their maximum value (denoted as iMax); when $\theta \leq -1$, all points are mapped to their minimum value (denoted as iMin). A 2-d example is given Fig.5, where $\theta = -1$ (iMin). The data points and the insertion order are as same as ones in Fig.1. To facilitate the KNN search, left pointers are added at the leaf nodes of the B^+-tree.

Orthogonal range queries on the original d-d data space have to be transformed to the union of d 1-d range queries. Given a query rectangle $W = [L_0, H_0] \times [L_1, H_1] \times \cdots \times [L_{d-1}, H_{d-1}]$, the orthogonal range search algorithm is given in Fig.6. Fig.7 shows the region visited when an orthogonal range search is performed in the iMinMax(θ).

```
PyramidRangeSearch($T, W$)
A ← $emptyset$
for ($i = 0; i < d; i \leftarrow i + 1$)
do $\overline{L}_i \leftarrow L_i - 0.5$
    $\overline{H}_i \leftarrow H_i - 0.5$
for ($i = 0; i < 2d; i \leftarrow i + 1$)
do $intersect \leftarrow 0$
    if ($i < d$)
       then for ($j = 0; j < d$ and $j \neq i; j \leftarrow j + 1$)
               do if ($\overline{L}_i \leq -MIN(\overline{L}_j, \overline{H}_j)$)
                     then $intersect \leftarrow intersect + 1$
               $q_{i_{min}} \leftarrow \overline{L}_i$
               $q_{i_{max}} \leftarrow min(\overline{H}_i, 0)$
       else for ($j = 0; j < d$ and $j \neq i - d; j \leftarrow j + 1$)
               do if ($\overline{H}_{i-d} \geq MIN(\overline{L}_j, \overline{H}_j)$)
                     then $intersect \leftarrow intersect + 1$
               $q_{i_{min}} \leftarrow max(\overline{L}_{i-d}, 0)$
               $q_{i_{max}} \leftarrow \overline{H}_{i-d}$
    if ($intersect = d - 1$)
       then $m \leftarrow 0$
               for ($j = 0; j < d; j \leftarrow j + 1$)
               do if ($\overline{L}_j \leq 0$) and ($\overline{H}_j \geq 0$)
                     then $m \leftarrow m + 1$
               if ($m = d$)
                  then $h_{low} \leftarrow 0$
                  else $q_{j_{max}} \leftarrow 0$
                       $q_{j_{min}} \leftarrow 0.5$
                       for ($j = 0; j < d$ and $j \neq i$ mod $d; j \leftarrow j + 1$)
                       do $q_{j_{max}} \leftarrow max(MIN(q_{i_{min}}, q_{i_{max}}), MIN(\overline{L}_j, \overline{H}_j))$
                          if ($q_{j_{min}} > q_{j_{max}}$)
                             then $q_{j_{min}} \leftarrow q_{j_{max}}$
                       $h_{low} \leftarrow q_{j_{min}}$
               $h_{high} \leftarrow max(|q_{i_{min}}|, |q_{i_{max}}|)$
               BplusTreeRangeSearch($T, i + h_{low}, i + h_{high}, W, A$)
return $A$
```

Figure 3: Orthogonal range search algorithm for the Pyramid technique, adapted from [4]. $MIN(a, b)$=0, if $a \leq 0 \leq b$, else $MIN(a, b) = min(|a|, |b|)$.

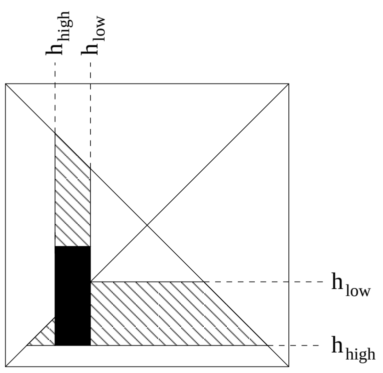


Figure 4: The data space and the query rectangle W (the black area is the region of W, and the cross-hatched area is the region needed to be visited during range search in addition to W) .

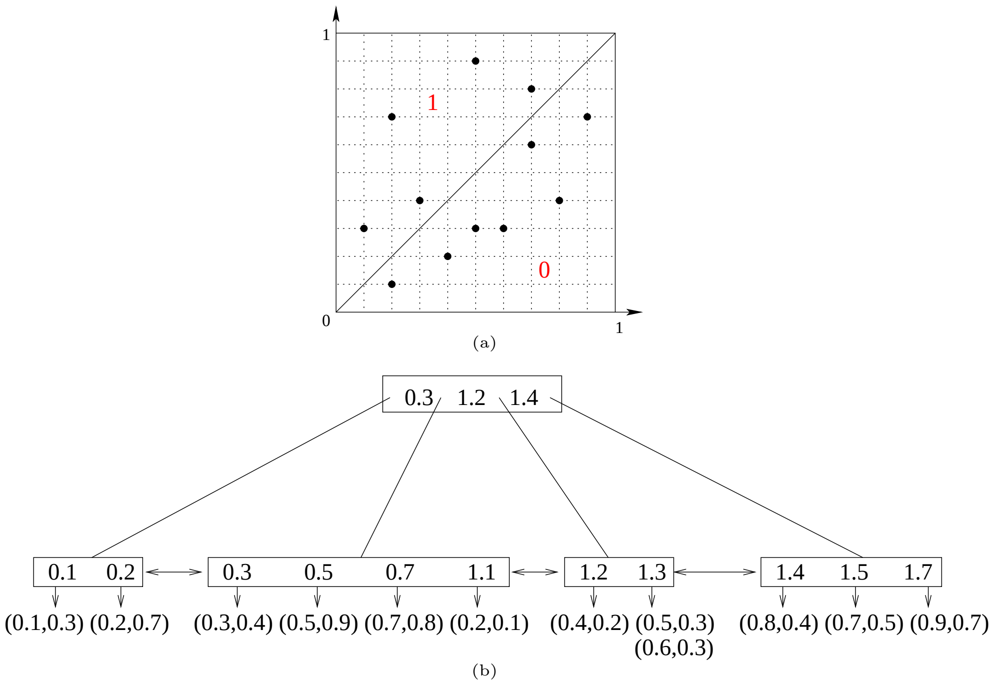
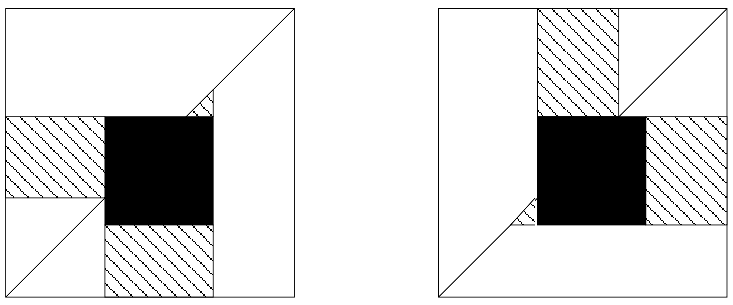


Figure 5: A 2-d iMin example. (a) A set of points in 2-d data space $[0, 1]^2$, which is divided by the diagonal $y{=}x$ to two right-angled triangles, numbered 0 and 1 (i.e. d_{min}), respectively. (b) the corresponding B^+-tree (the maximum number of keys is 4, and the point insertion order is (0.2,0.7), (0.1,0.3), (0.3,0.4), (0.2,0.1), (0.4,0.2), (0.5,0.3), (0.6,0.3), (0.8,0.4), (0.7,0.6), (0.9,0.7), (0.7,0.8), (0.5,0.9)).

```
iMinMaxRangeSearch($W$)
$A \leftarrow emptyset$
$L_{min} \leftarrow min_{i=0}^{d-1} L_i; L_{max} \leftarrow max_{i=0}^{d-1} L_i$
$H_{min} \leftarrow min_{i=0}^{d-1} H_i; H_{max} \leftarrow max_{i=0}^{d-1} H_i$
for $(i = 0; i < d; i \leftarrow i + 1)$
do if $(L_{min} + \theta \geq 1 - L_{max})$
      then $h_{low} \leftarrow L_{max}$
           $h_{high} \leftarrow H_i$
      else if $(H_{min} + \theta < 1 - H_{max})$
              then $h_{low} \leftarrow L_i$
                   $h_{high} \leftarrow H_{min}$
      else $h_{low} \leftarrow L_i$
           $h_{high} \leftarrow H_i$
   if $(h_{low} \leq h_{high})$
      then BplusTreeRangeSearch($T, i + h_{low}, i + h_{high}, W, A$)
return $A$
```

Figure 6: Range search algorithm for iMinMax(θ), adapted from [19].



(a) iMax (b) iMin

Figure 7: Sample search space in 2-d data space $[0, 1]^2$ (the black area is the region of the query rectangle W, and the cross-hatched area is the region needed to be visited during range search in addition to W) .

3 DR KNN Search Algorithms

To find the k nearest neighbors of a query point q, the distance of the kth neighbor from q defines the minimum radius required for searching k-nearest neighbors. Such a distance can't be determined in advance. The simplest KNN search algorithm is based on using a range search algorithm with increasing radius. For example, in [14][26], the KNN search starts with a query sphere centering at q with a small initial radius r. A candidate answer set is maintained which contains data points that could be the k nearest neighbors of q. Then the query sphere is enlarged gradually and the candidate answer set is updated accordingly until the k candidate answers are the true k nearest neighbors of q. Since the range search complexity grows sharply on the search radius, the cost of increasing radius approach can be very close to the cost of a range search with the appropriate r. The increasing radius method can be adapted to any data structure which supports range search. Fig.8 shows the increasing radius KNN algorithm in the Pyramid technique explored in [25].

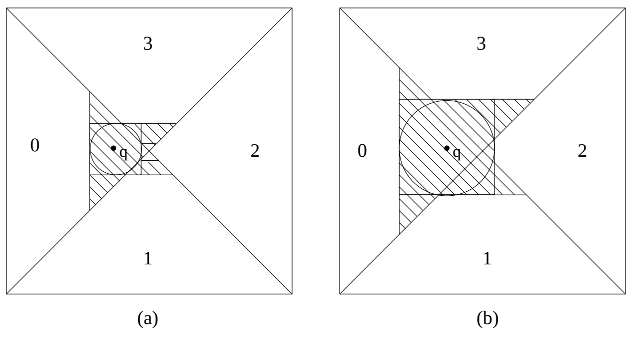


Figure 8: An illustration of a 2-d KNN search using increasing radius approach. (a) shows the KNN search with an initial radius, and (b) shows the KNN search with an increasing radius. The cross-hatched region is the search region.

3.1 DR Pyramid KNN Search Algorithm

In constrast to the KNN search algorithm with increasing radius (IR) for the Pyramid technique mentioned above [25], we present a KNN search algorithm with decreasing radius (DR). We use a list A to contain the k current candidate nearest neighbors sorted by their distance from q in decreasing order. Without loss of generality, we use the Euclidean distance (L_2). Let $D(v,q)$ be the Euclidean distance between points v and q in d-d space, and D_{max} be the maximum distance between the data points in A and q. Moreover, let $C(q,r)$ be a circle centered at q with a radius r.

In the first step, we initialize A to be empty, and determine which pyramid q is in and its pyramid value pv_q, using the algorithm in Fig.2. Assume q is in pyramid p_i, we search the B^+-tree to locate the leaf node which has the key value $= pv_q$, or the largest key value less than pv_q (using function LocateLeaf). After locating the leaf node, we use function SearchLeft (SearchRight) to check the data points of the node towards to the left (right) to determine if they are among the k nearest neighbors, and update A accordingly. Note that if a point v is in the same pyramid as q, the difference between their pyramid values is no greater than their Euclidean distance, i.e. $|pv_q - pv_v| \leq D(q,v)$. SearchLeft (SearchRight) stops when the key value of the leaf node is less (greater) than i (i+0.5), or there are k data points in A and the difference between the current key value in the node and the pyramid value of q is greater than D_{max}.

In the second step, since an exact circle shaped range search is hard to define in the Pyramid technique, we generate a query square W enclosing $C(q,r)$ to perform an orthogonal range search, which guarantees the correctness of the query results. If there are k data points in A, the radius r is initialized to be D_{max}, otherwise $r=\sqrt{d}$ such that $C(q,r)$ covers the whole data space $[0,1]^d$ (the maximum Euclidean distance between point v and point q in space $[0,1]^d$ is $\sqrt{d}$). For simplicity, we assume there are k data points in A after the first step. We examine the rest of the pyramids one by one in any order. If the pyramid intersects W, we perform a PyramidRangeSearch to check if the data points in this pyramid intersecting W are among the k nearest neighbors. The center of W is fixed, but its side length Δ is updated every time after a pyramid is examined. If the pyramid doesn't intersect W, we can prune the search in this pyramid. The KNN search stops when all the pyramids are checked. The KNN algorithm is given in Fig.9.

Fig.10 shows a KNN search example in 2-d data space. In this case, the number of the nearest neighbors k=4, and the data point q denoted as an unfilled circle is the query point. Firstly, we can determine that q is in pyramid p_0, then we search the B^+-tree using interval $[0, 0.5]$ to find the k candidate nearest neighbors of q in p_0, and store them in the list A, i.e. the

```
PyramidKNN($Point\ q$, $int\ k$)
 1  $A \leftarrow$ empty set
 2  $i \leftarrow$ pyramid number of the pyramid $q$ is in
 3  $node \leftarrow$ LocateLeaf($T, q$)
 4  SearchLeft($node, A, q, i$)
 5  SearchRight($node, A, q, i + 0.5$)
 6  $D_{max} \leftarrow D(A_0, q)$
 7  Generate $W$ centered at $q$ with $\Delta \leftarrow 2D_{max}$
 8  for ($j = 0; j < 2d; j \leftarrow j + 1$)
 9  do if ($j \neq i$) and ($W$ intersects pyramid $j$)
10        then PyramidRangeSearch($T, W, q, A$)
11             Update $W$ with updated $\Delta \leftarrow 2D_{max}$
12  return $A$
```

Figure 9: The decreasing radius Pyramid KNN search algorithm.

data points in $C(q, r)$ as shown in Fig.10(a), where r is the distance of the furthest data point from q in A (we assume there are k data points in A). Secondly, a query square W enclosing $C(q, r)$ is generated. We examine the rest of the pyramids in counterclockwise order. The cross-hatched area in pyramid p_1 in Fig.10(b) is the search region when we perform a 1-d range search on the B^+-tree. One candidate is found and A is updated, and so is r. The query square W is updated to enclose the updated $C(q, r)$. As a closer data point to q is found, we search with smaller radius and the search becomes cheaper. Because there is no intersection between W and pyramid p_2 (See Fig.10(c)), we don't need to check the data points in this pyramid. When we reach pyramid p_3, we examine the data points in cross-hatched region in Fig.10(d), and find that none of them is among the k-nearest neighbors. After checking 4 pyramids, we get the result, i.e. the data points except q in $C(q, r)$ in Fig.10(d).

3.2 DR iMinMax KNN Search Algorithm

The KNN search algorithm with decreasing radius for the iMinMax(θ) is given in Fig.7, which is similar to PyramidKNN algorithm in Fig.9. Firstly we initialize A to be empty, and calculate the iMinMax value θ_q of q. We search the B^+-tree to locate the leaf node which has the key value $= \theta_q$, or the

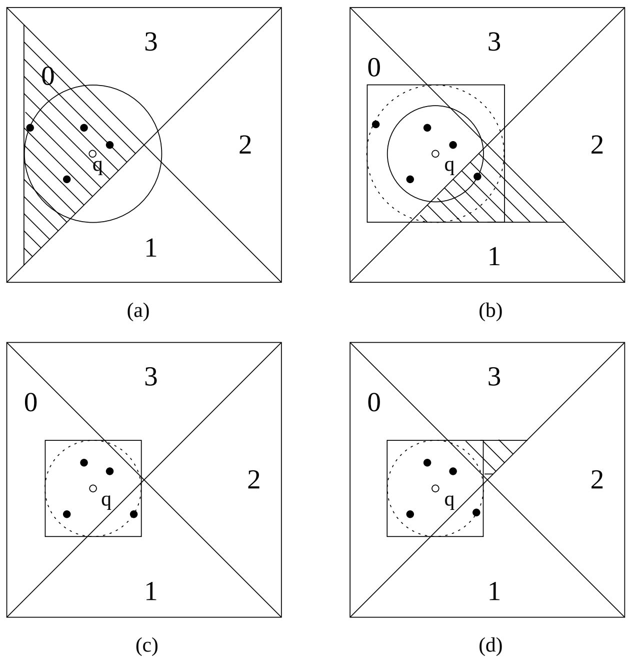


Figure 10: An illustration of a 2-d KNN query processing. The union of cross-hatched region in these four figures is the total search region for the KNN query.

largest key value $< \theta_q$. Then we use function SearchLeft (SearchRight) to check the data points of the node towards to the left (right) to determine if they are among the k nearest neighbors, and update A accordingly. Note that if a point v is in the same partition as q, the difference between their iMinMax values is no greater than their Euclidean distance, i.e. $|\theta_q - \theta_v| \leq D(q, v)$. SearchLeft (SearchRight) stops when the key value of the leaf node is less (greater) than θ_q (θ_q+1), or there are k data points in A and the difference between the current key value in the node and the iMinMax value of q is greater than D_{max}.

Secondly, we generate a query square W enclosing $C(q, r)$ to perform an orthogonal range search. We examine the rest of the partitions one by one in any order. We perform a iMinMaxRangeSearch to check if the data points in the partition intersecting W are among the k nearest neighbors. If there are some data points found in the partition whose distances less than D_{max}, A is updated. After finishing the search in this partition, r is reset to

```
iMinMaxKNN($Point\ q$, $int\ k$)
 1  $A \leftarrow$ empty set
 2  $i \leftarrow$ the number of the partition which $q$ is in
 3  $node \leftarrow$ LocateLeaf($T, q$)
 4  SearchLeft($node, A, q, i$)
 5  SearchRight($node, A, q, i+1$)
 6  $D_{max} \leftarrow D(A_0, q)$
 7  Generate $W$ centered at $q$ with $\Delta \leftarrow 2D_{max}$
 8  for ($j = 0; j < d$ and $j \neq i; j \leftarrow j+1$)
 9  do iMinMaxRangeSearch($T, W, q, A$)
10      Update $W$ with updated $\Delta \leftarrow 2D_{max}$
11  return $A$
```

Figure 11: The decreasing radius iMinMax KNN search algorithm.

the updated D_{max}. The search continues to examine one of the unexplored partitions. The KNN search stops when all the partitions are examined.

A 2-d DR iMinMax KNN search example is shown in Fig.12, where θ=-1 and k=4. Firstly, we can determine the query point q is in partition 1, then we search the B^+-tree using interval [1,2] to find the k candidate nearest neighbors in this right-angled triangle, stored in A, i.e. the data points in $C(q, r)$ as shown in Fig.12(a), where r=D_{max}. Secondly, we use a square enclosing $C(q, r)$ to perform an orthogonal range search on partition 0. The data points in solid circle in Fig.12(b) are the query results.

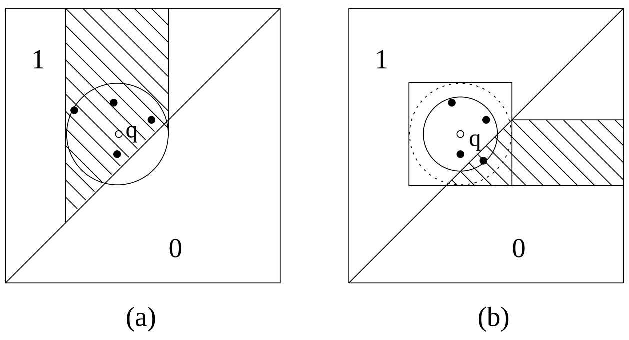


Figure 12: An illustration of a 2-d KNN query processing. The union of cross-hatched region in (a) and (b) is the total search region for the KNN query.

4 Experimental Evaluation

We have conducted a series of experiments to evaluate the DR KNN search performance. We first compared the increasing radius KNN search approach with the decreasing radius KNN approach using the Pyramid technique and the iMinMax(θ), then we compared the DR Pyramid and DR iMinMax KNN search performance with the R^*-tree, the BBD-tree and naive search. Our experiments were performed using uniformly and randomly distributed data points from the interval $[0,1]^d$, for $2 \leq d \leq 100$, and n up to 1,000,000. The programs were written in C++, and run on a Sun Microsystems V60 with two 2.8 GHz Intel Xeon processors and 3 GB main memory. We assume all data structures tested reside in the main memory, and we don't account for no I/O disk access. Each experimental point in the following graphs was done with an average of 300 queries which followed the same distribution of the data points.

4.1 DR vs. IR

For increasing radius KNN search, if the distance of the k-th nearest neighbor from the query point q (i.e. the minimum radius required for searching the k-nearest neighbors) can be determined in advance, the optimal IR KNN search performance can be achieved. However such a distance can't be predetermined. In our experiments, we set the initial radius of the range query as close to the minimum radius as possible. The volume of d-d ball with radius r is $\frac{\pi^{d/2} r^d}{\Gamma(\frac{1}{2}d+1)}$. For uniform and random data points, when searching for the k nearest neighbors, the expected volume of query ball is $\frac{k}{n}$. Then the initial radius $r{=}(\frac{k\Gamma(\frac{1}{2}d+1)}{n\pi^{d/2}})^{1/d}$.

Fig.13 and Fig.14 show the comparison between the IR KNN search and our DR KNN search performance for the Pyramid technique and the iMinMax(θ), respectively. The data points were drawn uniformly and randomly from d-d data space $[0,1]^d$. The performance of the iMinMax(θ) is affected by the value of θ, as shown in [19]. We set θ=0 in our experiments. The increasing radius Δr for each step in the IR KNN search was set to be $\frac{r}{m}$, where m is an integer. We varied m to examine the effect of the Δr in the IR KNN search performance. We observed that when d increases, the large Δr leads to a better performance, because there will be less iterations to reach the final query results. For the Pyramid technique, the DR KNN search has a speedup factor between 1.1 and 6.8 over the IR KNN search. For the iMinMax(θ), the DR KNN search has a speedup factor between 1.1 and 2.4 over the IR KNN search.

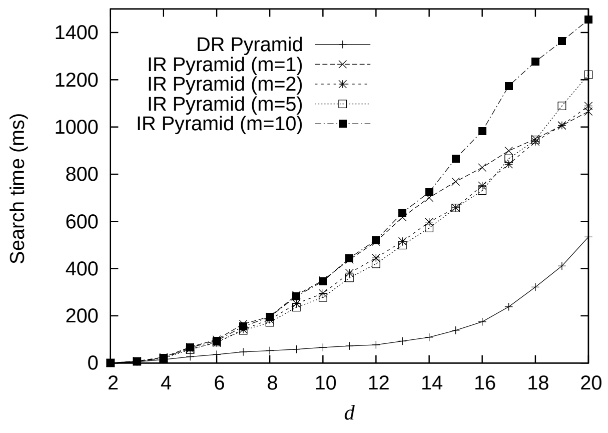


Figure 13: KNN search in the Pyramid technique, where n=1,000,000, 2≤ d ≤20, and k=10.

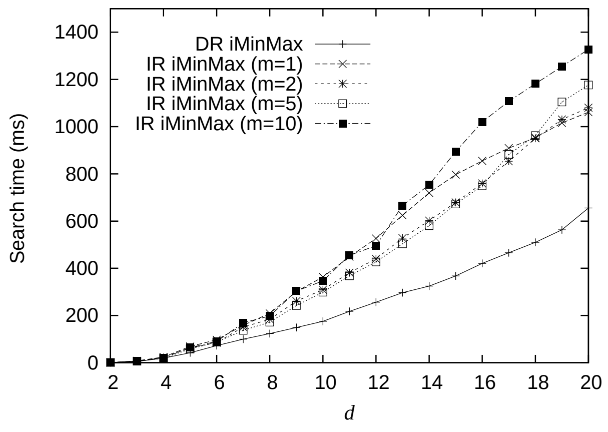


Figure 14: KNN search in the iMinMax(θ), where θ=0, n=1,000,000, 2≤ d ≤20, and k=10.

4.2 DR Pyramid Technique vs. Other Techniques

We compared the DR KNN performance of the Pyramid technique to the R^*-tree using the KNN algorithm in [21], the BBD-tree [2] and naive search. Naive search is a simple brute-force search. The BBD-tree was proposed for approximate KNN search, but it is able to return exact k-nearest neighbors when the error bound ϵ=0. The source code of the BBD-tree approximate KNN search is downloaded from [18]. To have a reasonable and fair comparison, we use the same uniform and random data points generator in all programs.

To determine the influence of the dimension d on KNN search performance, we varied d from 2 to 20. We fixed n=1,000,000, and k=10. In terms of search time, the experimental results in Fig.15 show that when $d < 8$, the DR Pyramid technique is slightly worse than the R^*-tree, and when $d \geq 8$, the DR Pyramid technique has a speedup factor up to 3.2 over the R^*-tree. The DR Pyramid KNN search outperforms the DR iMinMax KNN search. The BBD-tree is up to 13.4 times faster than the DR Pyramid technique; with increasing d, the KNN search performance for both data structures is approximately the same.

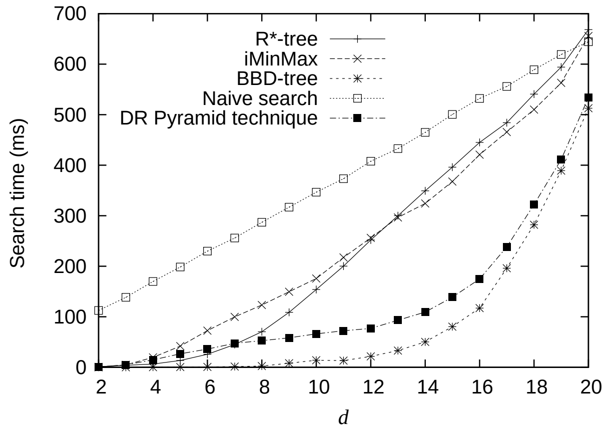


Figure 15: n=1,000,000, 2$\leq d \leq$20, and k=10.

Fig.16 shows the effect of k on the performance. k is varied from 5 to 50 stepping by 5. In Fig.16, d=16, the DR Pyramid technique has a speedup factor between 1.5 and 2.5 over the DR iMinMax, and a speedup factor between 1.6 and 3.0 over the R^*-tree. The BBD-tree has a speedup factor between 1.3 and 1.7 over the DR Pyramid technique. In the experiment of Fig.17, we examined the correlation between the input data size with the KNN search time. We varied the number n of data points from 100,000 to 1,000,000. The experimental results are shown in Fig.17, where k=20 and d=16. The KNN search time of the DR Pyramid technique, the BBD-tree, the R^*-tree and naive search increases linearly as n increases. The DR Pyramid technique has a speedup factor between 1 and 1.9 over the DR iMinMax, and a speedup factor between 1.3 and 2.2 over the R^*-tree. The BBD-tree is up to 1.2 times faster than the DR Pyramid technique.

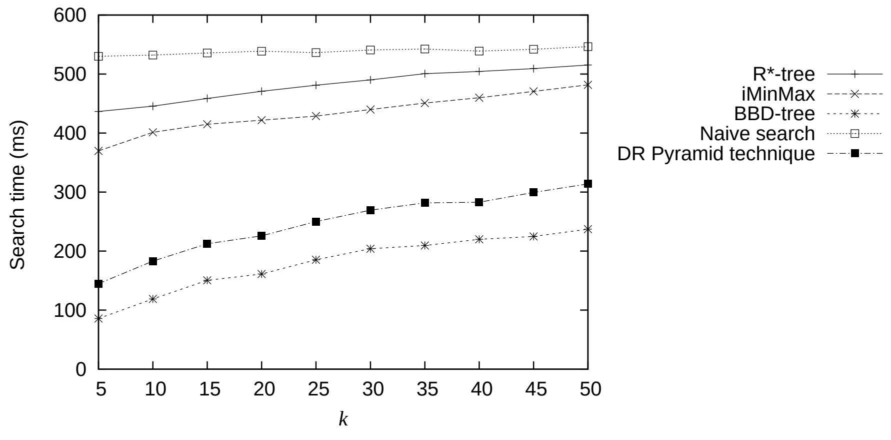


Figure 16: n=1,000,000, d=16, and 5≤k≤ 50.

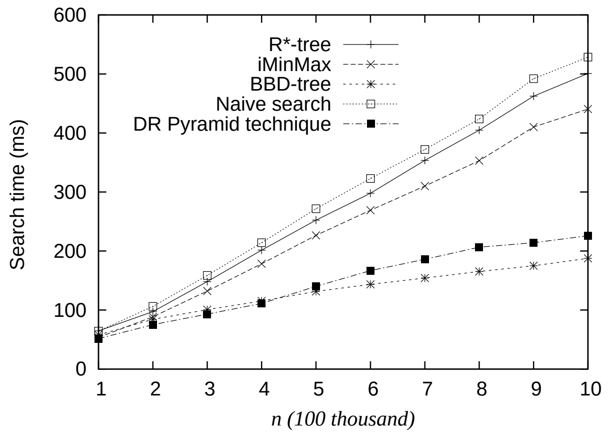


Figure 17: 100,000≤ n ≤1,000,000, d=16, and k=20.

5 Conclusions

In this report, we present a decreasing radius KNN algorithm for mapping-based indexing schemes. We implemented our DR KNN algorithm using the Pyramid technique and the iMinMax(θ), and conducted an extensive experimental evaluation to study the KNN search performance. The experiments show that the algorithm scales up well with both the number of nearest neighbors requested and the size of the data sets. For uniform random data points, the KNN performance of the DR Pyramid technique is faster than the IR Pyramid technique, the DR iMinMax and the R^*-tree. The BBD-tree is the best in all these data structures in experiments for $d \leq \log_2 n$. The Pyramid technique [4] and the iMinMax(θ) [19] have been shown to work well in high dimensional data spaces; can our DR KNN algorithm be improved to support efficient KNN search for large d ($d{>}\log_2 n$)? We somehow need to overcome the unnecessary space searched due to expanding the d-d query ball of radius r to a d-d query square of side length $2r$. The ratio between the volume ($(2r)^d$) of d-d square with side length $2r$ and the volume ($\pi^{d/2} r^d / \Gamma(\frac{1}{2}d + 1)$) of d-d ball with radius r is $\frac{\Gamma(\frac{1}{2}d+1)}{(\sqrt{\pi}/2)^d}$, which increases rapidly with the increment of d. For example, for d=2, the ratio≈1.27; for d=12, the ratio≈3067.48. What is the expected time $Q(k, d, n)$ for the KNN search using the DR Pyramid approach?

References

[1] S. Arya, D. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching. In *Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms*, pages 573–582, Arlington, VA, USA, January 23-25 1994.

[2] S. Arya, D. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, 45(6):891–923, 1998.

[3] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 322–331, Atlantic City, NJ, USA, May 23-25 1990.

[4] S. Berchtold, C. Böhm, and H.-P. Kriegel. The Pyramid-technique: Towards breaking the curse of dimensionality. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 142–153, Seattle, Washington, USA, June 2-4 1998.

[5] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree : An index structure for high-dimensional data. In *Proceeding of the 22nd International Conference on Very Large Data Bases*, pages 28–39, Bombay, India, September 3-6 1996.

[6] C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces–index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322–373, September 2001.

[7] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, September 2001.

[8] P. Ciaccia, M. Patella, and P. Zezula. M-trees: an efficient access method for similarity search in metric space. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 426–435, Athens, Greece, August 26-29 1997.

[9] B. Cui, B. C. Ooi, J. Su, and K.-L. Tan. Contorting high dimensional data for efficient main memory KNN processing. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 479–490, San Diego, CA, USA, June 9-12 2003.

[10] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3:231–262, July 1994.

[11] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. *Advances in Knowledge Discovery and Data Mining.* AAAI Press/MIT Press, Cambridge, Mass., 1996.

[12] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by image and video content: The QBIC system. *IEEE Computer*, 28(9):23–32, September 1995.

[13] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pages 47–57, Boston, MA, USA, June 18-21 1984.

[14] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang. iDistance: An adaptive B^+-tree based indexing method for nearest neighbor search. *ACM Transactions on Database Systems*, 30(2):364–397, June 2005.

[15] I. T. Jolliffe. *Principal Component Analysis.* Springer-Verlag, 2002.

[16] N. Katayama and S. Satoh. The SR-tree: An index structure for high dimensional nearest neighbor queries. In *Proceeding of the 1997 ACM-SIGMOD International Conference on Management of Data*, pages 369–380, Tucson, Arizona, USA, May 13-15 1997.

[17] R. Mehrotra and J. Gary. Feature-based retrieval of similar shapes. In *Proceeding of the 9th International Conference on Data Engineering*, pages 108–115, Vienna, Austria, April 19-23 1993.

[18] D. M. Mount and S. Arya. Ann: A library for approximate nearest neighbor searching. http://www.cs.umd.edu/ mount/ANN/, May 2005. Version 1.1.

[19] B. C. Ooi, K.-L. Tan, C. Yu, and S. Bressan. Indexing the edges - a simple and yet efficient approach to high-dimensional indexing. In *19th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 166–174, Dallas, Texas, USA, May 14-19 2000.

[20] H.-J. S. R. Weber and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24th International Conference on Very Large Data Bases*, pages 194–205, New York City, NY, USA, August 24-27 1998.

[21] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 71–79, San Jose, California, USA, May 22-25 1995.

[22] S. Shekhar, S. Chawla, S. Ravada, and A. Fetterer. Spatial databases - accomplishments and research needs. *IEEE transactions on knowledge and data engineering*, 11(1):45–55, 1999.

[23] D. A. White and R. Jain. Similarity indexing with the SS-tree. In *Proceeding of the 12th International Conference on Data Engineering*, pages 516–523, New Orleans, Louisiana, USA, February 26-March 1 1996.

[24] E. Wold, T. Blum, D. Keislar, and J. Wheaton. Content-based classification, search, and retrieval of audio. *IEEE Multimedia*, 3(3):27–36, 1996.

[25] R. Zhang, P. Kalnis, B. C. Ooi, and K.-L. Tan. Generalized multidimensional data mapping and query processing. *ACM Transactions on Database Systems*, 30(3):661–697, September 2005.

[26] R. Zhang, B. C. Ooi, and K.-L. Tan. Making the pyramid techique robust to query types and workloads. In *Proceedings of the 20th International Conference on Data Engineering*, pages 313–324, Boston, USA, March 30-April 2 2004.