K-Nearest Neighbor Search using the Pyramid Technique

Bradford G. Nickerson Qingxiu Shi *

Abstract

A k-nearest neighbor search algorithm for the Pyramid technique is presented. The Pyramid technique divides d-dimensional data space into $2d$ pyramids. 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 which q is in, then examine the rest of the pyramids one by one. After one pyramid is checked, the radius of the range query decreases or remains the same. We compared the k-nearest neighbor performance of the Pyramid technique with the k-d tree, the R^*-tree and naive search. Experimental results show that our decreasing radius Pyramid k-nearest neighbor search algorithm is efficient when $d \leq \log_2 n$.

1 Introduction

K-nearest neighbor search is an important problem in computational geometry. Given a set S of n data points and a query point q, 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). K-nearest neighbor search is used in many applications, such as computer graphics, multimedia databases, geographic information systems, knowledge discovery and data mining, and computer aided design systems [5]. Several k-nearest neighbor algorithms have been proposed [3].

To find the k nearest neighbors of a query point q, the maximum distance of the k-nearest neighbors from q defines the minimum radius required for searching k-nearest neighbors. Such a distance can't be determined in advance. Zhang et al. [7] employed an increasing radius approach: the search starts with a query sphere centering at q with a small initial radius. They maintain a candidate answer set 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.

In contrast to the k-nearest neighbor search approach with increasing radius (IR) for the Pyramid technique explored in [7], we present a k-nearest neighbor search algorithm with decreasing radius (DR) in this paper. The Pyramid technique divides the data space $[0,1]^d$ into $2d$ pyramids. The radius r of a range query is initialized to be the distance of the furthest data point in the k candidate nearest neighbors after examining the data points in the pyramid 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. 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, and the radius r decreases if at least one new candidate is found. The process is repeated until all pyramids are examined. Though the algorithm is designed for the Pyramid technique, it is applicable to other mapping-based indexing schemes [7].

2 The Pyramid Technique

The basic idea of the Pyramid technique [2] is to transform the d-dimensional (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.

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 $i \bmod d$, i.e. $h_v = |0.5 - v_{i \bmod d}| \in [0, 0.5]$. 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)).

Given a query rectangle W, an orthogonal range search finds the points intersecting W. The Pyramid technique maps a d-d range query into a union of 1-

*University of New Brunswick, Fredericton, NB, E3B 5A3, Canada {bgn,l749u}@unb.ca

d range queries. First we determine which pyramids are intersected by W. Then we determine which pyramid values inside an intersected pyramid p_i intersect W. When we reach the leaf level of the B^+-tree, we examine if the data points storing in the leaf nodes intersect W. Fig.3 shows the region visited when an orthogonal range search is performed.

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, 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.

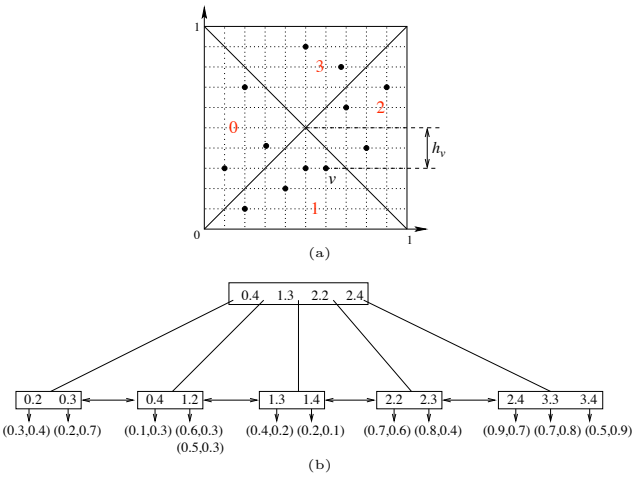

Figure 1: (a) A set of points in 2-d data space $[0,1]^2$ (the numbers in the triangles are the pyramid numbers i), and (b) the corresponding B^+-tree (the maximum number of keys is 4, and the point insertion order is (0.2,0.7), (0.1,0.3), (0.3,0.4), (0.2,0.1), (0.4,0.2), (0.5,0.3), (0.6,0.3), (0.8,0.4),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

3 K-Nearest Neighbor Search Algorithm

Given a query point q, a k-nearest neighbor search finds the k data points in the data set closest in distance to q. 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. Let $D(v,q)$ be the Euclidean distance between point v and point 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

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

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

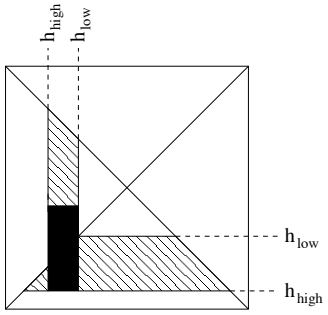

Figure 3: The data space and the query rectangle W (the black area is the region of W, and the cross-hatched area is the region needed to be visited during range search in addition to W) .

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, we generate a query square W of side length Δ=2r 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 (using the Intersection algorithm

```
KNN(Point q , int k )
 1  A ← empty set
 2  i ← pyramid number of the pyramid q is in
 3  node ← LocateLeaf(T, q)
 4  SearchLeft(node, A, q, i)
 5  SearchRight(node, A, q, i + 0.5)
 6  D_max ← D(A_0, q)
 7  Generate W centered at q with Δ ← 2D_max
 8  for (j = 0; j < 2d; j ← j + 1)
 9  do if (j ≠ i) and (W intersects pyramid j )
10        then RangeSearch(T, h_low, h_high, W, q, A)
11             Update W with updated Δ ← 2D_max
12  return A
```

Figure 4: The decreasing radius Pyramid k-nearest neighbor search algorithm. T is the root of the B^+-tree. h_{low}=j+h^j_{low} and h_{high}=j+h^j_{high} (h^j_{low} and h^j_{high} are determined by the Interval algorithm [6]).

in [6]), after determining h_{low} and h_{high} using the Interval algorithm [6], we perform a RangeSearch 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 k-nearest neighbor search stops when all the pyramids are checked. The k-nearest neighbor algorithm is given in Fig.4.

Fig.5 shows a k-nearest neighbor 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 data points in $C(q, r)$ as shown in Fig.5(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.5(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.5(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.5(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.5(d).

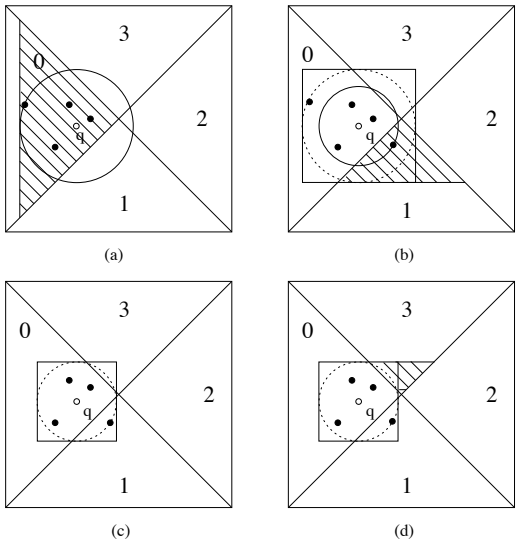

Figure 5: An illustration of a 2-d k-nearest neighbor query processing. The union of cross-hatched region in these four figures is the total search region for the k-nearest neighbor query.

4 Experimental Evaluation

We conducted experiments to demonstrate the efficiency of our approach. We examined the behavior of our algorithm by varying the number of neighbors, the dimension of data space and the input data set size using synthetic data. Data points were drawn from a uniformly and randomly distributed space $[0, 1]^d$. We compared the k-nearest neighbor search performance of the Pyramid technique to the k-d tree, the R^*-tree using the k-nearest neighbor algorithm in [4] and naive search. Naive search is a simple brute-force search. In the following figures, the DR Pyramid technique uses the algorithm in Fig.4 to perform the k-nearest neighbor search, and the IR Pyramid technique and the k-d tree use the increasing radius approach [3]. The radius of a range query is initialized to be $(\frac{k\Gamma(\frac{1}{2}d+1)}{n\pi^{d/2}})^{1/d}$, which is the radius of the query ball expected to contain k points, and increased gradually until the k nearest neighbors are found.

To determine the influence of the dimension d on k-nearest neighbor search performance, we varied d from 2 to 20. We fixed the data set size n=1,000,000, and the number of the nearest neighbors k=10. In terms of search time spent, the experimental results in Fig.6 show that the DR Pyramid technique has a speed-up factor up to 2.7 over the IR Pyramid technique, 2.9 over the k-d tree, and 3.8 over the R^*-tree. Fig.7 shows the effect of k on the performance. k is varied from 5 to 50 stepping by 5. In Fig.7, d=16, the DR Pyramid technique is up to 2.1 over the IR Pyramid technique, 1.8 over the k-d tree and 3.2 times over the R^*-tree.

In the experiment of Fig.8, we examined the correlation between the input data size with the k-nearest neighbor search time. We varied the number n of the data points from 100,000 to 1,000,000. The experimental results are shown in Fig.8, where k=20 and d=16. The k-nearest neighbor search time of the DR Pyramid technique, the IR Pyramid technique, the k-d tree, the R^*-tree and naive search increases linearly as n increases, but the rate of increment for the DR Pyramid technique is slowest. The Pyramid technique DR has a speed-up factor 1.9 over the IR Pyramid technique, 1.6 over the k-d tree and 2.4 over the R^*-tree. 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. Each experimental point in the graphs was done with an average of 300 test cases.

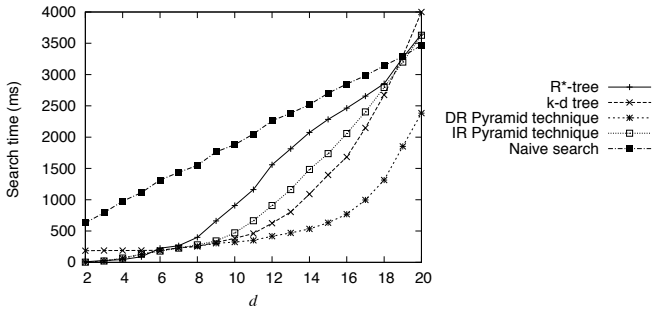

Figure 6: n=1,000,000, 2$\leq d \leq$20, and k=10.

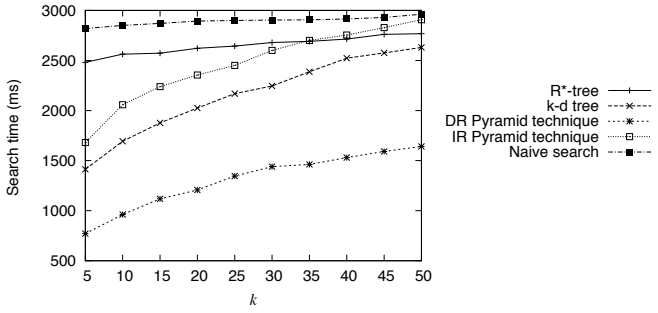

Figure 7: n=1,000,000, d=16, and 5$\leq k \leq$ 50.

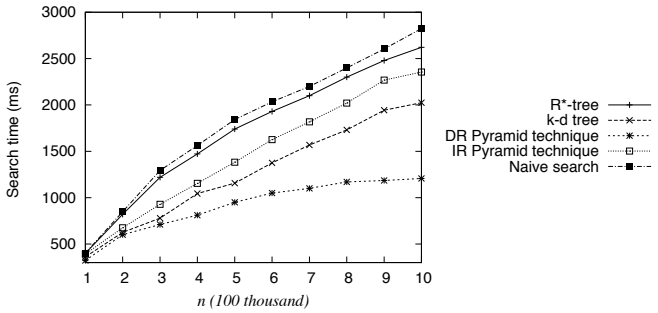

Figure 8: 100,000$\leq n \leq$1,000,000, d=16, and k=20.

5 Conclusions

In this paper, we present a decreasing radius k-nearest neighbor algorithm for the Pyramid technique. We implemented our DR Pyramid algorithm and conducted an extensive experimental evaluation to study the k-nearest neighbor search performance of the DR Pyramid technique. 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 k-nearest neighbor performance of the DR Pyramid technique is faster than the k-d tree, the R^*-tree and naive search. The Pyramid technique [2] has been shown to work well in high dimensional data spaces; can our DR Pyramid algorithm be improved to support efficient k-nearest neighbor 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 k-nearest neighbor search using the DR Pyramid approach? Is the DR Pyramid approach competitive for approximate k-nearest neighbor search [1]?

References

[1] 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.

[2] 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.

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

[4] 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.

[5] 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.

[6] Q. Shi and B. G. Nickerson. On k-d range search with large k. Technical report, TR06-176, Faculty of Computer Science, University of New Brunswick, May 2006, 25 pages.

[7] 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.