The PKD-tree for Orthogonal d-dimensional Range Search

Bradford G. Nickerson Qingxiu Shi *

Abstract

We present two new d-dimensional data structures, called the PKD-tree and the PKD^+-tree, respectively. They are explored for indexing combined text and point data in low and high dimensional data spaces, and evaluated experimentally for orthogonal range search (for $2 \leq d \leq 128$ and n up to 1,000,000) using synthetic data points and real data. The experimental results show that the PKD-tree and the PKD^+-tree work well for any d, and they always outperform the Pyramid technique, and are greatly better than the k-d tree and the R^*-tree when $d \geq \log_2 n$. For a PKD^+-tree built from n uniform and random data points, an orthogonal range search with a query square W of side length Δ visits $O(d \log n + n(1-(1-2\Delta)^d))$ nodes for $\Delta \leq 0.5$.

1 Introduction

Given a collection of n records, each containing d attributes, a range search asks for all records in the collection with key values inside a specified range for each of the d dimensions. Over the past 30 years, more than 100 data structures for the range search problem have been presented [2][6].

The traditional indexing schemes, e.g. the k-d tree and the quadtree, divide d-dimensional (d-d) data space into subspaces using (d-1)-d hyperplane parallel to the coordinate axes. They perform efficiently on small d ($d \leq \log_2 n$), but when d is getting larger, they suffer from the curse of dimensionality. 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 [1] and the iMinMax(θ) [5]. The basic idea is to transform the d-d data points into 1-d values, and then store and access the 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. [8] proposed a generalized structure for multi-dimensional data mapping and query processing. The mapping-based indexing scheme overcomes the high dimensionality problem. However when $d \leq \log n$, the mapping-based indexing method tends to examine more data points during range search than the traditional indexing schemes, e.g. the k-d tree.

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

We can take advantage of the traditional indexing schemes for small d ($d \leq \log_2 n$), and the superiority of the mapping-based indexing scheme for $d > \log_2 n$. In this paper, we combine the Pyramid technique and the k-d tree, called the PKD-tree and the PKD^+-tree, respectively. In Sections 2 and 3, we propose these two data structures, and theoretically analyze their space requirement and range search time in the worst case. We present the experimental results from an extensive performance study to evaluate the PKD-tree for orthogonal range search in Section 4, using synthetic data points and real data. Overall, our experiments show that the PKD-tree and the PKD^+-tree are better than the Pyramid technique when $d \leq \log_2 n$, and they approximate the Pyramid technique when $d >> \log_2 n$. The PKD-tree and its variant outperform the k-d tree and the R^*-tree when $d \geq \log_2 n$ and don't deteriorate with increasing d.

2 The PKD-tree

Given a d-d key $v = (v_0, v_1, \cdots, v_{d-1})$, the attribute v_j can be a point coordinate value or a text string, $0 \leq j \leq d-1$. We map the coordinate values to $[0,1]$. When v_j is a text string, we use the numeric mapping method [4] to get a numeric value in $[0,1]$. We denote the mapped d-d key as $\overline{v} = (\overline{v}_0, \overline{v}_1, \cdots, \overline{v}_{d-1})$, $\overline{v}_j \in [0,1]$, $0 \leq j \leq d-1$.

We use the Pyramid technique [1] to transform the d-d data points to 1-d values. The maximum distance h_v from v to the center point $(0.5, 0.5, \cdots, 0.5)$ is defined to be $h_v = \max_{j=0}^{d-1} |0.5 - \overline{v}_j|$, with j_{max}= dimension where h_v is found. We say v is located in pyramid i, and $i = j_{max}$ if $\overline{v}_{j_{max}} < 0.5$, or $i = d + j_{max}$ if $\overline{v}_{j_{max}} \geq 0.5$. The pyramid value pv_v of v is $pv_v = i + h_v$. The algorithm for calculating pv_v is given in [7].

We denote by T the PKD-tree constructed by inserting n keys into an initially empty tree. The root of the PKD-tree is an internal node with $2d$ child pointers, indexing $2d$ pyramids respectively. After determining which pyramid v is in, we insert v into the corresponding B^+-tree using pv_v as a key. When we reach the leaf level of the B^+-tree, we insert v into the associated k-d tree (See Fig.1). Let S be the maximum number of points in the k-d tree. If the number of points in the k-d tree is S before inserting v, pv_v is inserted into the B^+-tree as a key, and the k-d tree is partitioned into two k-d trees according to the key values in the B^+-tree. As we

know that the k-d tree is relatively slow for large d, we define $S = \sqrt{\frac{n}{2d}}$ such that each k-d tree has fewer points when d is large. This means more of the search is done in the pyramid part of the PKD-tree, which is more efficient for large d. The challenge for the PKD-tree is to determine an appropriate S such that an excellent range search performance of the PKD-tree is always achieved. It is obvious that the space requirement of the PKD-tree is $O(dn)$. In the following discussions, we assume the B^+-tree has order M. The internal nodes of a B^+-tree of order M contains between M and $2M$ keys, and a node with m keys has $m+1$ children.

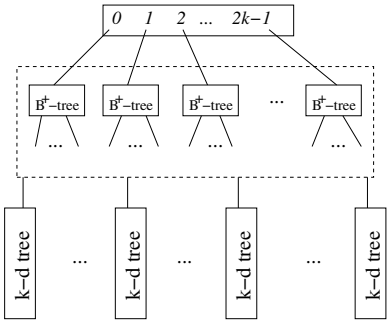


Figure 1: The data structure of PKD-tree.

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 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 $\overline{H}_i = H_i - 0.5$. Range search begins from the root of the PKD-tree. If the pyramid i $(0 \le i \le 2d-1)$ is intersected by W (INTERSECTION algorithm in Fig.2), we visited the B^+-tree which the ith child the root points to using interval $[i + h_{low}[i], i + h_{high}[i]]$ ($h_{low}[i]$ and $h_{high}[i]$ are determined by INTERVAL algorithm in Fig.3. We define a more restricted value of $h_{low}[i]$ than the one in [1]). When we reach the leaf level of the B^+-tree, we visit the associated k-d tree to determine if the data points inside W: starting at the root of the k-d tree, at each node v, we compare the v_j attribute of the current node with $[L_j, H_j]$ $(0 \le j < d$, j is the current node's discriminator). If $v_j \le L_j$, the search continues on the left child of v; if $v_j > H_j$, the search continues on the right child of v; otherwise, we check if v inside W and the search continues on both children of v.

Theorem 1 *Given a PKD-tree built from n d-d data points drawn from a uniform random distribution $[0,1]^d$, and a query square W with side length Δ, an orthogonal range query visits*

1. $O(d \log \frac{n}{dS} + \frac{n(1-(1-2\Delta)^d)}{S^{1/d}} + F)$, *if* $\Delta \le 0.5$
2. $O(d \log \frac{n}{dS} + \frac{n(1+(2\Delta-1)^d)}{S^{1/d}} + F)$, *if* $\Delta > 0.5$

nodes in the PKD-tree, where F is the number of data points in range.

INTERSECTION($\overline{W}$)

```
1  Initialize boolean array intersect[2*d]←0
2  for (i = 0; i < d; i ← i + 1)
3  do x ← 0; y ← 0
4     for (j = 0; j < d and j ≠ i; j ← j + 1)
5     do if (L̄_i ≤ (−MIN(L̄_j, H̄_j)))
6           then x ← x + 1
7        if (H̄_i ≥ MIN(L̄_j, H̄_j))
8           then y ← y + 1
9     if (x = d − 1)
10       then intersect[i] ← 1
11    if (y = d − 1)
12       then intersect[d + i] ← 1
```

Figure 2: Algorithm determining which of the $2d$ pyramids are intersected by W, adapted from [1].

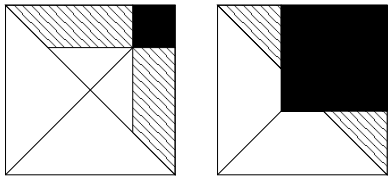
INTERVAL($\overline{W}$)

```
1  for (i = 0; i < 2d; i ← i + 1)
2  do if (i < d)
3        then q_{i_min} ← L̄_i; q_{i_max} ← min(H̄_i, 0)
4        else  q_{i_min} ← max(L̄_{i−d}, 0); q_{i_max} ← H̄_{i−d}
5     m ← 0
6     for (j = 0; j < d; j ← j + 1)
7     do if (L̄_j ≤ 0) and (H̄_j ≥ 0)
8           then m ← m + 1
9     if (m = d)
10       then h_low[i] ← 0
11       else  q_{j_max} ← 0; q_{j_min} ← 0.5
12             for (j = 0; j < d and j ≠ i  mod d;
13                  j ← j + 1)
14             do q_{j_max} ← max(MIN(q_{i_min}, q_{i_max}),
15                                MIN(L̄_j, H̄_j))
16                if (q_{j_min} > q_{j_max})
17                   then q_{j_min} ← q_{j_max}
18             h_low[i] ← q_{j_min}
19    h_high[i] ← max(|q_{i_min}|, |q_{i_max}|)
```

Figure 3: Algorithm for determining h^i_{low} and h^i_{high} in each pyramid i. $MIN(a,b) = 0$, if $a \le 0 \le b$, else $MIN(a,b) = \min(|a|, |b|)$.

Proof. We assume the input data points are randomly and uniformly distributed. For an input size n, the number of data points in each B^+-tree is expected to be $\frac{n}{2d}$, or $O(\frac{n}{d})$, then the number of k-d trees in each B^+-tree of order M is $O(\frac{n}{dS})$, and the height of the B^+-tree of order M is $O(\log_{M+1} \frac{n}{dMS})$. The worst case happens when W is in the corner of the data space (See Fig.4).

When $\Delta \le 0.5$, the volume of pyramid i to be visited intersecting W is $\frac{1}{2d} - \frac{(2(0.5-\Delta))^d}{2d} = \frac{1}{2d}(1-(1-2\Delta)^d)$. There are $\frac{n}{2d}(1-(1-2\Delta)^d)$ data points in the shadow region of pyramid i. The number of the k-d trees associative to the B^+-tree for the pyramid i to be visited is $\lceil \frac{n}{2dS}(1-(1-2\Delta)^d) \rceil + 1$. The worst case of the number of nodes visited in the k-d tree having S nodes during range search is $O(S^{(1-1/d)} + f)$, where f is the number



$\Delta \leq 0.5$ $\Delta > 0.5$

Figure 4: Illustration of the worst case for a PKD-tree range search. The black area corresponds to query square W, and the cross-hatched area is the region visited during the range search in addition to W.

of nodes in range in the k-d tree. The number of leaf nodes in the B^+-tree visited is $\lceil \frac{\lceil \frac{n}{2dS}(1-(1-2\Delta)^d)\rceil+1}{M} \rceil$, and the number of internal nodes visited is the height of the B^+-tree. There are d pyramids intersecting W, so the total nodes visited in the PKD-tree is thus $O(d \log_{M+1} \frac{n}{dMS} + \frac{n(1-(1-2\Delta)^d)}{S^{1/d}} + F)$.

In a similar way, when $\Delta > 0.5$, the center of the data space is contained in W, so all pyramids are intersected by W. The total volume to be visited is $d(\frac{1}{2d} + \frac{(2(\Delta-0.5))^d}{2d}) = \frac{1}{2}(1 + (2\Delta-1)^d)$, The maximum number of nodes visited in the PKD-tree is thus $O(2d \log_{M+1} \frac{n}{dMS} + \frac{n(1+(2\Delta-1)^d)}{S^{1/d}} + F)$. □

3 The PKD^+-tree

The basic structure of the PKD^+-tree is a B^+-tree of order M. The pyramid values of data points are used as the key value in the B^+-tree. An internal node with m keys of the PKD^+-tree has one right pointer, $m+1$ child pointers and $m+1$ associated k-d tree pointers, each child pointer is related to a k-d tree (denoted as KD) pointer. The data points stored in the leaf nodes of the subtree which the child pointer pointed to are store in the nodes of the k-d tree (e.g. in Fig. 5, the leftmost KD of the root contains (0.3,0.4) and (0.2,0.7), and the rightmost KD has (0.9,0.7), (0.7,0.8) and (0.5,0.9)). The leaf node of the PKD^+-tree with m keys has one right pointer and m data point pointers. The right pointer points to the immediate right node at the same level. The orthogonal range search algorithm is given in [7].

Theorem 2 *The PKD^+-tree of order M built from n d-d data points requires $O(dn \log n)$ space.*

Proof. The height of the B^+-tree of order M is $O(\log_{M+1} n)$. The number of the nodes in the B^+-tree is $O(\frac{n}{M})$, which require $O(n)$ space. The leaf nodes need additional dn space for the data points. At each level above the leaf node level, there are totally n nodes in all associated KDs, which require dn space. So the total storage of the PKD^+-tree is $O(dn \log_{M+1} n)$. □

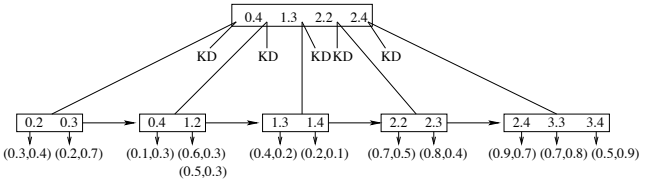


Figure 5: A 2-d PKD^+-tree example (the point insertion order is (0.2,0.7), (0.1,0.3), (0.3,0.4), (0.2,0.1), (0.4,0.2), (0.5,0.3), (0.6,0.3), (0.8,0.4), (0.7,0.5), (0.9,0.7), (0.7,0.8), (0.5,0.9)).

Theorem 3 *Given a PKD^+-tree built from n d-d data points drawn from a uniform random distribution $[0,1]^d$, and a query square W with side length Δ, an orthogonal range search query visits*

1. *$O(d \log n + n(1-(1-2\Delta)^d))$, if $\Delta \leq 0.5$*
2. *$O(d \log n + n(1+(2\Delta-1)^d))$, if $\Delta > 0.5$*

nodes in the PKD^+-tree.

Proof. The worst case happens when the range search doesn't search any KD and checks the data points in leaf nodes, and W is in the corner of the data space, sharing a vertex and d edges with the space (See Fig.4). The height of the B^+-tree of order M is at most $\lfloor \log_{M+1} n \rfloor$.

When $\Delta \leq 0.5$, the volume of pyramid i to be visited intersecting W is $\frac{1}{2d} - \frac{(2(0.5-\Delta))^d}{2d} = \frac{1}{2d}(1-(1-2\Delta)^d)$. There are d pyramids intersecting W, so the total volume to be visited is $\frac{1}{2}(1-(1-2\Delta)^d)$. For uniformly and randomly distributed points, there are at most $\frac{1}{2}n(1-(1-2\Delta)^d)/M$ leaf nodes in the B^+-tree, and $\frac{1}{2}n(1-(1-2\Delta)^d)$ data points pointed by the leaf nodes visited. With the number of internal nodes visited in the B^+-tree, the maximum number of nodes visited in the PKD^+-tree is thus $d \log_{M+1} n + \frac{M+1}{2M} n(1-(1-2\Delta)^d)$.

When $\Delta > 0.5$, the center of the data space is contained in W, so all pyramids are intersected by W. The total volume to be visited is $d(\frac{1}{2d} + \frac{(2(\Delta-0.5))^d}{2d}) = \frac{1}{2}(1 + (2\Delta-1)^d)$, so there are at most $\frac{1}{2}n(1+(2\Delta-1)^d)/M$ leaf nodes in the B^+-tree, and $\frac{1}{2}n(1+(2\Delta-1)^d)$ data points pointed by the leaf nodes visited. The maximum number of nodes visited in the PKD^+-tree is thus $2d \log_{M+1} n + \frac{M+1}{2M} n(1+(2\Delta-1)^d)$. □

4 Experiments

The programs were run on a Sun Microsystems V880 with four 1.2 GHz UltraSPARC III processors, 16 GB of main memory, running Solaris 8. We assume all data structures tested reside in the main memory, and there is no I/O disk access. Each experimental point in the following graphs was done with an average of 300 test cases.

4.1 Synthetic Data

Data points were drawn uniformly and randomly from $[0,1]^d$. To determine the influence of the dimension on range search performance, we vary d from 2 to 128. We fix the input point data size n=100,000, and the expected output $E(F)$=$\log_2 n$. For the constant F, the query rectangle W varies according to d. The experimental results in Fig.6 show that for low value of d<16, the k-d tree takes about 0.7 of the search time required by the PKD-tree, with the R^*-tree a close second. When $d\geq$16, the PKD-tree is up to 10.4 times faster than the k-d tree and 5.4 times faster than the R^*-tree. The PKD-tree is up to 13.5 times faster than the Pyramid technique.

In Fig.7, we measured range search performance behavior with the input data size n varying from 100,000 to 1,000,000. $E(F)$ is set to be $(\log_2 n)^2$, and d is 16. The PKD-tree has speed-up factor up to 3.9 over the k-d tree, 3.6 over the Pyramid technique and 2.4 over the R^*-tree. The PKD^+-tree is close to the PKD-tree in Fig. 6 and Fig.7.

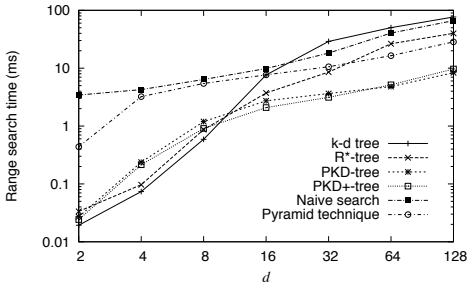


Figure 6: n=100,000, 2$\leq d\leq$128, and $E(F)$=$\log_2 n$.

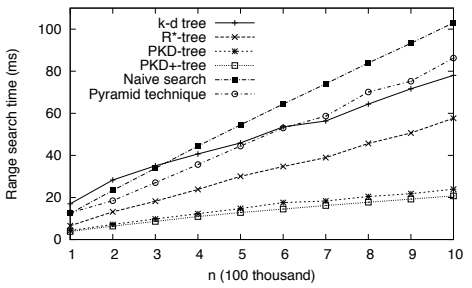


Figure 7: $E(F)$=$(\log_2 n)^2$, 100,000$\leq n\leq$1,000,000 and d=16.

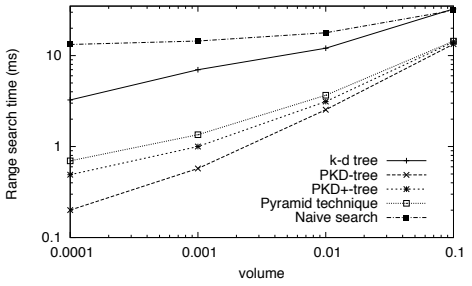


Figure 8: $volume$=Δ^d=$E(F)/n$, n=68,040 and d=32.

4.2 Real Data

We tested the data structures on a color histogram data set [3], which has 68,040 32-dimensional data points on $[0,1]^{32}$. In Fig.8, the volume of the query square W with side length Δ ($volume$=Δ^d=$E(F)/n$) is varied from 0.0001 to 0.1. The PKD-tree has a speed-up factor between 2.4 ($volume$=0.1) and 16 ($volume$=0.0001) compared to the k-d tree, between 1.1 and 3.4 compared to the Pyramid technique. The PKD-tree is slightly faster than the PKD^+-tree.

5 Conclusions

We propose the PKD-tree and the PKD^+-tree for orthogonal range search in low and high dimensional data space. We conducted an extensive experimental study to evaluate their range search performance, and compared them to the k-d tree, the Pyramid technique, the R^*-tree and naive search. Overall, the experimental results show the PKD-tree and its variant work well for any d ($2 \leq d \leq 128$). They are always better than the Pyramid technique, and outperform the k-d tree and the R^*-tree when $d \geq \log_2 n$. What is the expected range search time of the PKD-tree and the PKD^+-tree?

References

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

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

[3] S. Hettich and S. D. Bay. *The UCI KDD Archive*. http://kdd.ics.uci.edu, Department of Information and Computer Science, University of California, Irvine, CA, 1999.

[4] H. Jagadish, N. Koudas, and D. Srivastava. On effective multi-dimensional indexing for strings. In *Proc. ACM SIGMOD Int. Conf. on Management of data*, pages 403–414, Dallas, Texas, USA, May 14-19 2000.

[5] 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.

[6] H. Samet. *The design and analysis of spatial data structures*. Computer Science Department, University of Maryland, College Park, Maryland, USA, 2004.

[7] 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.

[8] 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.