Chapter 1

PK-TREE: A SPATIAL INDEX STRUCTURE FOR HIGH DIMENSIONAL POINT DATA

Wei Wang
and Jiong Yang
IBM T.J. Watson Research Center
{ww1, jiyang}@us.ibm.com

Richard Muntz
Department of Computer Science
University of California, Los Angeles
muntz@cs.ucla.edu

Abstract In this chapter we present the PK-tree which is an index structure for high dimensional point data. The proposed indexing structure can be viewed as combining aspects of the PR-quad or K-D tree but where unnecessary nodes are eliminated. The unnecessary nodes are typically the result of skew in the point distribution and we show that by eliminating these nodes the performance of the resulting index is robust to skewed data distributions. The index structure is formally defined, efficiently updatable and bounds on the number of nodes and the mean height of the tree can be proved. Bounds on the expected height of the tree can be given under certain mild constraints on the spatial distribution of points. Empirical evidence both on real data sets and generated data sets shows that the PK-tree outperforms the recently proposed spatial indexes based on the R-tree such as the SR-tree and X-tree by a wide margin. It is also significant that the relative performance advantage of the PK-tree grows with the dimensionality of the data set.

Keywords: multi-dimensional index, spatial indexing, point data sets, image database.

1. INTRODUCTION

Large databases of objects are becoming more important in applications such as medical image archives, biology, CAD, etc. It is common in such databases for objects to be described by a vector of numeric attributes. Particularly in similarity search applications, e.g., in content based access in image databases, high dimensional spaces are common. Therefore, a variety of dynamic spatial indexing structures have been proposed in the past few years [Abr95, Bec90, Ber96, Cia97, Gut84, Hen89, Kam94, Kat97, Lin94, Rob81, Sam90, Sel87][1]. Most index structures become less efficient with increasing dimensionality and this is where the challenge lies.

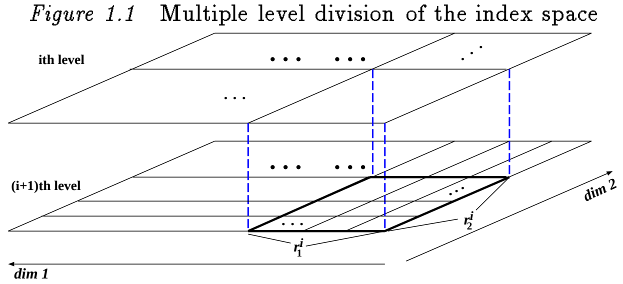
Many applications can benefit from the simple and easy tree traversal property of spatial decomposition. We introduce a new spatial indexing structure based on regular spatial decomposition, the PK-tree (Pyramid K-instantiable tree). A PK-tree achieves a better bound on the height of the tree for skewed data distributions than other trees based on regular decomposition because it only instantiates non-leaf cells with at least a certain number of nonempty children. This restriction largely eliminates the problem of the height of the tree growing large due to skew in the spatial distribution of points. Updates to the PK-tree are inexpensive and the performance of the PK-tree is shown via experiments to compare well with that of any existing variations of the R-tree for the most common types of queries especially for higher dimensional data. Moreover, a PK-tree is independent of the order of data insertions and deletions. Last but not least, the PK-tree is based on a solid theoretical foundation. Its uniqueness for any data set, a bound on the mean height, and bounds on the number of children in a node can all be proved formally [Yan97].

The remainder of the chapter is organized as follows. We formally define the PK-tree structure and present the properties of the PK-tree in Section 2. Then in Section 3, the operations of the PK-tree is introduced. In Section 4, we summarize the results of empirical studies comparing PK-tree performance with other spatial index methods. Finally, we summarize and discuss conclusions in Section 5.

2. DEFINITION OF PK-TREE

In this section, we formally define the structure of a PK-tree. Assume a given set of m-dimensional points such that all points are in a given rectangular area[2], C_0. Consider a recursive rectilinear[3] division of C_0 into successively smaller rectangular cells. We divide the cell at level 0 into several disjoint subcells at level 1, these cells at level 1 are each divided into disjoint subcells at level 2, and so on. The division is independent of the data set. The recursive subdivision used in constructing the PR-quadtree [Sam90] is an example of this kind of recursive division. Division on each level could be different. For example, one can divide on dimension 1 and 2 at level i and divide on dimension 3 at level $i+1$. For the spatial area C_0, we thus have different sets of cells at different levels, with cells at level $i+1$ nested within cells at level i. The higher the level is, the smaller the cell size is. The length of the side of a cell along dimension j at level i is denoted by L_j^i, and $\forall i > 0$, $L_j^i = \frac{L_j^{i-1}}{r_j^i}$ where $r_j^i \geq 1$ is a constant integer. We call r_j^i the dividing ratio of level i along dimension j. For the same level, the dividing ratios along different dimensions may be different. For instance, if each cell at level i is trisected along dimension 1 but not dimension 2, then $r_1^i = 3$ and $r_2^i = 1$. Let $R^i = (r_1^i, r_2^i, \ldots, r_m^i)$ and $R = \cup_i\{R^i\}$ be the dividing ratio tuple at level i and the overall dividing ratio tuple respectively, where m is the number of dimensions. We also require that $\forall i > 0$, $r_1^i \times r_2^i \times \ldots \times r_m^i > 1$, i.e., a cell has to be partitioned on at least one dimension at each level. Let $r_i = r_1^i \times r_2^i \times \ldots \times r_m^i > 1$ be the fanout factor at level i. Then each cell at level i is always divided into r_i cells at level $i+1$. Let $r_{max} = \max_i(r_i)$ and $r_{min} = \min_i(r_i)$ be the maximum and minimum fanout factor over all levels, respectively. Since all $r_j^i > 1$ are constant integers, r_{max} and r_{min} are also constant integers and $r_{max} \geq r_{min} \geq 2$. The cells at two successive levels are illustrated in Figure 1.1 for the two dimensional case.

Figure 1.1 Multiple level division of the index space



Looking ahead, the PK-tree will have nodes, each of which corresponds to a cell at some level. The parent-child relationship corresponds naturally to the cell-subcell relationship. Let L_D be the minimum level such that all points in a set D are in separate cells. If all cells to level L_D are included in such a tree, the number of nodes is very large and the structure will tend to be inefficient. The PR-quadtree [Sam90] does not include subtrees of nodes that contain only one point (or less than some specified number of points). However, the non-leaf nodes that remain can have as little as one child. The "sparsely filled" nodes occupy space and increase the cost of queries unnecessarily. A PK-tree basically applies a simple rule to eliminate nodes with relatively few children from the tree in such a way that the result is an efficient tree structure with a number of desirable properties, including uniqueness independent of the order of insertions and deletions, bounded number of children, non-overlapping of sibling cells and a bound on the average height.

The concept of **K-instantiable cells** is fundamental to the definition of PK-trees. We begin with some basic definitions which lead, by the end of this section, to the definition of K-instantiable cells and the PK-tree.

For any data point d, we use $(d_1, d_2, \ldots, d_m)$ to denote its location where d_j is the coordinate of d along dimension j. Let C_0 be the rectangular region containing all possible data points, $C_0 = \{d | \forall j (1 \leq j \leq m) : l_j \leq d_j < u_j\}$ where l_j and u_j are the predefined minimum and maximum coordinates along dimension j. We call C_0 the *index space.*

Definition 1.1 *Given an index space C_0, a finite set of points D and dividing ratio tuple R, let L_D denote the minimum level such that each cell at level L_D contains at most one point (or multiple "collocated" points). Each cell at level L_D which contains a point will be referred to as a* point cell.

Definition 1.2 *Let C be a cell in m dimensional space and $[C_1^{min}, C_1^{max}]$, $[C_2^{min}, C_2^{max}], \ldots, [C_m^{min}, C_m^{max}]$ be the projection of this cell onto each dimension, where $C_j^{min} \leq C_j^{max}$, $\forall j = 1, 2, \ldots, m$. For any data point $d \in C_0$, we say $d \in C$ iff $C_j^{min} \leq d_j < C_j^{max}$, $\forall j = 1, 2, \ldots, m$. Otherwise, $d \notin C$.*

Each cell can be viewed as the set of points contained in it and set notation is used to express relationships among cells. For example, we say $C_1 \subseteq C_2$ iff $\forall d \in C_0 (d \in C_1 \Longrightarrow d \in C_2)$. We also say that C_1 is a sub-cell of C_2 (or C_2 is a super-cell of C_1) if $C_1 \subseteq C_2$ and C_1 is a proper sub-cell of C_2 (or C_2 is a proper super-cell of C_1) if $C_1 \subset C_2$. Let D be the set of all data points in an application, $D \subset C_0$. The cardinality of D is finite and denoted by N.

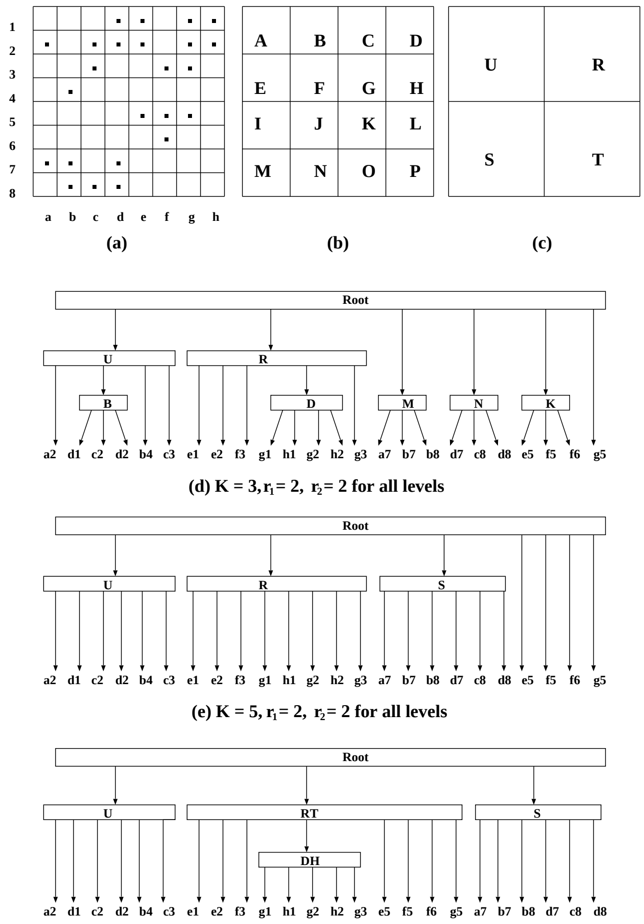
The following definition is not easy to grasp intuitively but an example is given immediately following the definition to illustrate the concept.

Definition 1.3 *Given a finite set of points $D \subset C_0$ and a dividing ratio tuple R, a cell C is K-instantiable ($K > 1$) iff*

1. *C is a point cell, or*
2. *$\nexists$ $K - 1$ or less K-instantiable cells $C_1, \ldots, C_{K-1} \subset C$, such that $\forall d \in D (d \in C \Longrightarrow d \in \bigcup_{i=1}^{K-1} C_i)$.*

Consider the example illustrated in Figure 1.2. In this case the space is two dimensional, the dividing ratios are $r_1^i = 2$ and $r_2^i = 2$ for all levels in Figure 1.2(d)(e), and we choose to set $K = 3$ in Figure 1.2(d) and $K = 5$ in Figure 1.2(e). In Figure 1.2(a) $L_D = 3$ and the cells containing the "dots" are the point cells which, by definition, are all K-instantiable for any K. Of the cells at level two, A is not 3-instantiable since the points contained in A can be covered by only one subcell (namely $a2$) but B is 3-instantiable since the points it contains cannot be covered by less than three 3-instantiable subcells. In Figure 1.2(f), the tree for the same data set is built with $K = 5$ and the dividing ratios $r_1^i = 2$ and $r_2^i = 1$ when i is odd and $r_1^i = 1$ and $r_2^i = 2$ when i is even. It follows from the

Figure 1.2 Examples of PK-trees



(a) (b) (c)

(d) K = 3, r_1 = 2, r_2 = 2 for all levels

(e) K = 5, r_1 = 2, r_2 = 2 for all levels

(f) K = 5, r_1 = 2, r_2 = 1 for odd levels and r_1 = 1, r_2 = 2 for even levels definition that the following theorem holds[4].

Theorem 1 *For a given area C_0, data set D, and dividing ratio tuple R, the set of K-instantiable cells is unique for any given value of K.*

Definition 1.4 *Given a finite set of points D over index space C_0 and dividing ratio tuple R, a PK-tree of rank K $(K > 1)$ is defined as follows.*

1. *The cell at level 0 (corresponding to C_0) is always instantiated and serves as the root of the PK-tree.*
2. *Except for the root, every node in the PK-tree is mapped one-to-one to a K-instantiable cell.*
3. *For any two nodes C_1 and C_2 in the PK-tree, C_1 is a (immediate) child of C_2 (or C_2 is the parent of C_1) iff*
   (a) *C_1 is a proper sub-cell of C_2, i.e., $C_1 \subset C_2$, and*
   (b) *$\nexists$ node C_3 in the PK-tree such that $C_1 \subset C_3$ and $C_3 \subset C_2$.*

Theorem 2 (Existence and Uniqueness) *For a given finite set of points D of cardinality N, in an index space C_0, and dividing ratio tuple R, there exists a unique PK-tree of rank K $(K > 1)$.*

Theorem 3 (Bounds on Node Outdegree) *For a given index space C_0 and maximum fanout factor r_{max}, each intermediate node in a PK-tree of rank K $(K > 1)$ has at least K and at most $(K-1) \times r_{max}$ children.*

Theorem 4 (Bounded Storage Space) *Any PK-tree of rank K for a finite set of points D, of cardinality N, index space C_0, and maximum fanout factor r_{max} has at least $N + \frac{N-1}{(K-1)\times r_{max}-1}$ and at most $N + \frac{N+K-2}{K-1}$ nodes.*

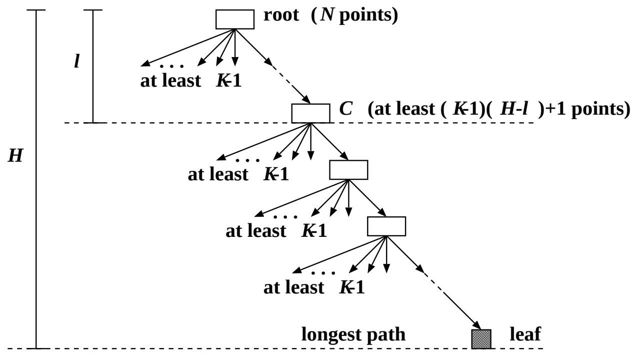
Expected Height of a PK-tree

Let $D = \{x_1, x_2, \ldots, x_N\}$ be a random set of N points in a space C_0. $P_D(x_1, x_2, \ldots, x_N)$ is the probability distribution for D. We construct the PK-tree T from D and denote the height of T by $h(T)$. For a given C_0 and R, let $\wp(C_0, K, R)$ be the set of cells in the corresponding pyramid structure. Let C_1, C_2, …, C_i, … be a *cell sequence* where $C_1 \supset C_2 \supset \ldots \supset C_i \supset \ldots$ in which C_i is a cell at level i of $\wp(C_0, K, R)$. Let $P_i = P(d \in C_{i+1} \mid d \in C_i)$ be the probability that a random point $d \in D$ is in C_{i+1} given that d is in C_i. Given a *cell* *sequence* $C_1, C_2, \ldots$, the corresponding *path sequence* ρ is $C_1, C_2, \ldots, C_\pi$ where π is the smallest integer such that $P_\pi = 0$ if there exists such a π; otherwise, the corresponding path sequence is the cell sequence. Clearly, if $P(d \in C_{\pi+1} \mid d \in C_\pi) = 0$, then $C_{\pi+1}$ and all its subcells are empty and hence are not K-instantiable. Given a path sequence $\rho = C_1, C_2, \ldots$, the corresponding *sub-path sequence* ρ' is a subsequence[5] of ρ such that each C_i is an element in ρ' iff $P_i < 1$. The motivation for defining the sub-path sequence is to eliminate from the path the cells that would not be K-instantiable; any cell for which $P_i = 1$ only contains points in one subcell at the next level and is never K-instantiable. For a sub-path sequence ρ', let $\psi_{\rho'} = \max(P_i)$ for all cells C_i in ρ' be the probability bound on ρ'. Let $\Psi = \max(\psi_{\rho'})$ be the overall probability bound of all sub-path sequence ρ'. (For example, for a uniform spatial distribution of points, $\Psi = \frac{1}{r_{min}}$). To simplify the discussion here, we assume that there exists a small constant $\epsilon > 0$ such that $\Psi \leq 1 - \epsilon$. However, all results still hold without this assumption. The proof of the general case can be found in [Yan97].

Property 1.1 *Let D be a set of N points as defined above with parameter $\Psi \leq 1 - \epsilon$ and T be the PK-tree for D with rank K. Let $h(T)$ be a random variable equal to the height of T. Then $P(h > H) = O\left(\Psi^{\log^2 N}\right)$ where $H = \Omega(\log N)$, $H > 4\log N$ and Ψ is the probability bound defined above.*

Property 1.2 *The expected height of the PK-tree for a set D of N data points with parameter $\Psi \leq 1 - \epsilon$ is $O(\log N)$.*

Figure 1.3 Expected Height of PK-tree



Intuitively, for any node C on the longest path of the PK-tree, if the height of the subtree rooted at C is $H - l$, then there are at least

$(H-l)(K-1)+1$ leaf nodes in such a subtree. The reason is that there are at least $(H-l)$ intermediate nodes in such a subtree and each intermediate node has at least K children. Then for each node on the longest path, there are at least $K-1$ out-branches besides the one in the longest path. Since each out-branch has at least one leaf node as its descendant, there must be at least $(H-l)(K-1)+1$ leaf nodes in this subtree. Therefore, at least $(H-l)(K-1)+1$ data points have to be located within C in order that the height of the tree is H. This is illustrated in Figure 1.3.

Since there are at least l nodes on the path from the root to C and $\Psi \leq 1-\epsilon$ is the upper bound of the probability $P(d \in C_{k1} \mid d \in C_{k2})$ where $C_{k1} \subset C_{k2}$ are two consecutive cells in a sub-path sequence, the probability of a data point to be in C is at most Ψ^l. If we choose H and l such that $l > 3\log N$ and $H - l \geq \log N$, by Chernoff Bound [Mot97], the probability that there exists such a cell C containing at least $(H-l)(K-1)+1$ points decreases exponentially as H increases.[6]

Since from the definition, it is not intuitive how general $\Psi \leq 1-\epsilon$ is, the above results can be difficult to apply directly in a real situation. However, we are developing simpler, sufficient conditions (such as multi-level clustering spatial distributions and bounded intensity constraint) to guarantee $\Psi \leq 1-\epsilon$ (and therefore, $E(h) = O(\log N)$) that should be more easily verified in practice. Interested readers can refer to [Wan97] for detail.

3. OPERATIONS ON A PK-TREE

Let d denote the data point that is to be inserted into T or deleted from T and let C_d be the point cell that contains d. The insertion of a data point d into the PK-tree T is achieved by inserting the corresponding leaf node C_d into T. This can be done in two phases: the first phase is to follow the path from the root T down to the leaf level to locate all (potential) ancestors of C_d while the second phase is from the leaf level back to the root along the same path to make all necessary changes (instantiate or deinstantiate cells) to maintain the tree as a PK-tree. The formal description of the insertion algorithm is in [Wan97]. The computational complexity of data point insertion is linearly proportional to the height of the PK-tree. (Since the number of children of each node is bounded, the complexity of the algorithm at each level is constant.) Since the average height of the PK-tree is $O(\log N)$, the average computational complexity is $O(\log N)$ and the worst case complexity is $O(N)$ where N is the number of data points in the PK-tree. The deletion procedure is similar to insertion, which is also presented in [Wan97].

By appropriate choice of parameters the index structure can be allocated one node per page achieving acceptable storage utilization as well as low I/O counts per query. In addition, a consistency control scheme is designed to allow multiple users to access the index concurrently even when multiple writers are active. Due to the space limitation, we will not present both issues in this chapter. Interested readers are referred to [Wan98] for details.

K-nearest-neighbor query and range query are common types of queries operated on a spatial index. Since these query algorithms are similar to those used in PR-Quad trees, we will not discuss them here. Both algorithms are in [Wan97]. In addition, the PK-tree can be generated from sorted data efficiently ($O(N)$ complexity). Due to the limit of space, we will not present this efficient loading algorithm in this chapter. Interested readers can refer to [Yan97].

4. PERFORMANCE COMPARISON OF PK-TREE, SR-TREE, AND X-TREE

4.1 EXPERIMENTAL SETUP

All the performance tests in this section were done on a SPARC-10 workstation (SunOS 5.5) with 208 MB main memory and a local disk with 9GByte capacity. We chose to compare PK-tree with X-tree and SR-tree because the X-tree and SR-tree are among the most recently proposed dynamic spatial indexing structures and they are reported to have significant performance improvement over R*-tree, K-D-B tree, and so on [Kat97][Ber96]. For the K-Nearest-Neighbor query and Range query, we choose to use no cache for PK-tree, SR-tree, X-tree. We use the version of X-tree source code given to us by the authors of [Ber96] recently, and the SR-tree source code from the authors of [Kat97].

For the PK-tree, in order to fit one node in one 8 KB disk block and keep the utilization level high, we fix the dividing ratio to be 2 at all levels, and round-robin through the dimensions, one dimension at each level. We vary the rank of the PK-tree for different dimensionality so that the maximum number of children, i.e. $2(K-1)$, will fill an 8 KB page as tightly as possible. In these performance tests, we are mainly concerned with K-Nearest-Neighbor (KNN) and Range Queries. We choose $K = 25$ for all KNN queries while we vary the distance in the range query so that around 10 points will be included.

In order to demonstrate the performance of PK-tree, we use both synthetic and real data sets. Each synthetic data set consists of one hundred thousand data points. The results of the performance tests are discussed in the following subsections.

4.2 SYNTHETIC DATA SET

Skewed data distributions are common place in many real applications. To explore this issue, the data is generated as follows: 20% of data are uniformly distributed in space and 80% of data are distributed in fifteen disjoint clusters. Within each cluster, the coordinate of data points along each dimension is normally distributed with the mean at the center of the cluster and the standard deviation 0.025. The heights of the PK-tree, X-tree, and SR-tree are listed in Table 1.1 while the size of each index is listed in Table 1.2. The PK-tree only instantiates the set of nodes with at least K children, and hence it is guaranteed that each disk block has at least 50% utilization when the dividing ratio on each level is chosen as 2. (The average disk utilization level was actually observed to be well above 80% for the three synthetic and one real data sets.)

Table 1.1 The Height (maximum levels) of Indexes with 100,000 points

Dimension	*2*	*4*	*8*	*16*	*32*	*64*
PK-tree	7	7	6	7	8	9
X-tree	4	4	4	4	5	6
SR-tree	4	4	5	5	6	7

Table 1.2 The size of Indexes in Megabytes with 100,000 points

Dimension	*2*	*4*	*8*	*16*
PK-tree	1.9	2.8	4.9	9.4
X-tree	1.8	3.0	5.6	10.6
SR-tree	70	74	75	92

Figures 1.4 and 1.5 show the performance of the KNN and range queries, respectively. If a KNN query is performed on a point within a cluster, then many points outside the cluster are quickly eliminated. If a KNN query is performed on a point outside any cluster, then depending on the data location, the performance may vary significantly. We randomly chose 20 points for each query. The average number of I/O and CPU time are low, for all three indexes, compared with uniformly distributed data set. Since the PK-tree employs disjoint spatial division, it can take most advantage of this kind of clustered data set and it has the lowest expected number of I/O and average CPU time.

Figure 1.4 KNN Query Performance for Clustered Data Distribution

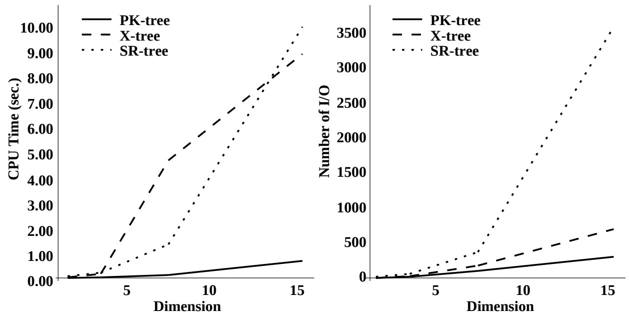
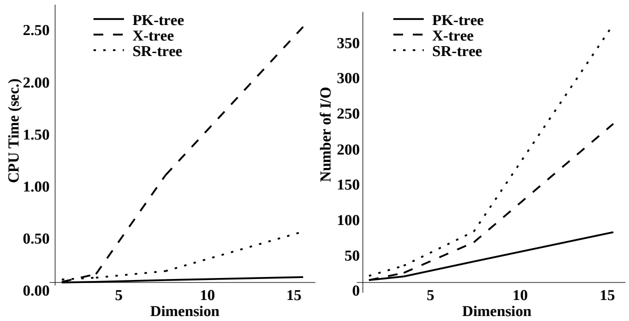


Figure 1.5 Range Query Performance for Clustered Data Distribution



4.3 REAL DATA SET

We chose the NASA Sky Telescope data as an example real data set. The NASA data set consists of 200,000 two-dimensional points (they are the coordinates of crater locations on the surface of Mars). Table 1.3 shows the average tree height, index structure size, the average CPU time, the expected number of I/O of KNN and Range query. Based on

Table 1.3 The Performance on NASA Sky Telescope Data Set

	height	*size*	*KNN CPU*	*KNN I/O*	*RAN CPU*	*RAN I/O*
PK-tree	5	3.7MB	4ms	4	3ms	4
X-tree	4	5.7 MB	90ms	4	10ms	4
SR-tree	5	120MB	28ms	8	14ms	6

the test results presented above, we can see that PK-tree outperforms the X-tree and SR-tree by a large margin, especially in high dimensionalities because the overlap among sibling nodes in an X-tree or an SR-tree increases significantly with dimensionality. On the other hand, there is no overlap among any sibling nodes in a PK-tree.

5. SUMMARY

In this chapter, we propose a new dynamic spatial indexing structure, called the PK-tree. In some sense, it can be conceived as a variation of PR quad-trees. However, it differs from all existing trees by employing a unique set of constraints to eliminate unnecessary nodes that can result from a skewed spatial distribution of objects. This ensures that the total number of nodes in a PK-tree is $O(N)$ and the average height of a PK-tree is $O(\log N)$ under some general conditions. In addition, the PK-tree has a set of nice properties include: non-overlapping of sibling nodes, uniqueness of the PK-tree for a given set of data points, and so on. Empirical studies shown that the PK-tree outperforms SR-tree and X-tree by a wide margin. Therefore, we see the PK-tree as a preferable indexing method.

Notes

1. Due to the space limitations, we will not elaborate them in this chapter. Interested reader please refer to the individual papers.
2. If the spatial area of interest is not a rectangular shape, then C_0 is the minimum bounding rectangle of the area of interest. The development can be also easily extended to the case that the spatial area of interest is dynamically defined as data points are added, but due to space limitations, we will not address this issue here.
3. This is for simplicity of exposition. Many of the properties of the PK-tree are independent of this assumption.
4. We omit all proofs in this chapter due to space limitation. Interested reader please refer to [Yan97]
5. Given a sequence, a subsequence contains a subset of elements from the original sequence in the same order, but not necessarily consecutive in the original sequence.
6. The complete formal proofs of the above properties are in [Yan97].

References

[Abr95] M. Abrash. BSP Trees. *Dr. Dobbs Sourcebook*, 20(14), 49-52, May/June 1995.

[Bec90] N. Beckmann, H.-P. Kriegel, R. Schneider, and Bernhard Seeger. The R*-tree: an efficient and robust access method for points and rectangles. *Proc. ACM SIGMOD Conf. on Management of Data*, 322-331, 1990.

[Ber96] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree : an index structure for high-dimensional data. *Proc. 22nd Int. Conf. on Very Large Data Bases (VLDB)*, 28-39, 1996.

[Cia97] P. Ciaccia, M. Patella, and P. Zezula. M-tree: an efficient access method for similarity search in metric spaces. *Proc. 23rd Int. Conf. on Very Large Data Bases (VLDB)*, 426-435, 1997.

[Gut84] A. Guttman. R-trees: a dynamic index structure for spatial searching. *Proc. ACM SIGMOD Conf. on Management of Data*, 47-57, 1984.

[Hen89] A. Henrich, H.-W. Six, and P. Widmayer. The LSD tree: spatial access to multi-dimensional point and non-point objects. *Proc. 15th Int. Conf. on Very Large Data Bases (VLDB)*, 45-54, 1989.

[Kam94] I. Kamel and C. Faloutsos. Hilbert R-tree: an improved R-tree using fractals. *Proc. 20th Int. Conf. on Very Large Data Bases (VLDB)*, 500-509, 1994.

[Kat97] N. Katayama and S. Satoh. The SR-tree: an index structure for high-dimensional nearest neighbor queries. *Proc. ACM SIGMOD Conf. on Management of Data*, 369-380, 1997.

[Lin94] K.-I. Lin, H. V. Jagadish, and C. Faloutsos. The TV-tree: an index structure for high-dimensional data. *VLDB Journal*, 3(4):517-542, 1994.

[Mot97] R. Motwani. *Randomized Algorithms*, Cambridge University Press, 1997.

[Rob81] J. T. Robinson. The K-D-B-tree: a search structure for large multidimensional dynamic indexes. *Proc. ACM SIGMOD Conf. on Management of Data*, 10-18, 1981.

[Sam90] H. Samet. The design and analysis of spatial data structures. *Addison-Wesley Publishing Company*, 1990.

[Sel87] T. K. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-tree: a dynamic index for multi-dimensional objects. *Proc. 13th Int. Conf. on Very Large Data Bases (VLDB)*, 507-518, 1987.

[Wan97] W. Wang, J. Yang, and R. Muntz. PK-tree: a dynamic spatial index structure for large data sets. *UCLA Computer Science Department Technical Report #970039*, 1997.

[Wan98] W. Wang, J. Yang, and R. Muntz. PK-tree: a spatial index structure for high dimensional point data. *UCLA Computer Science Department Technical Report #980032*, 1998.

[Wan98b] W. Wang, J. Yang, and R. Muntz. PK-tree: a spatial index structure for high dimensional point data. *Proc. Int. Conf. on Foundations of Data Organozation and Algorithms (FODO)*, 1998.

[Yan97] J. Yang, W. Wang, and R. Muntz. Yet another spatial indexing structure. *UCLA Computer Science Department Technical Report #970040*, 1997.