Location based hybrid indexing structure - R k-d Tree

P.AnandhaKumar,J.Priyadarshini,C.Monisha, K.Sugirtha, Sandhya Raghavan

Department of Information Technology, MIT, Anna University

Email: anandh@annauniv.edu,Vinodha_priya@yahoo.com,sandhyasan1988rag@gmail.com,k.sugirtha@gmail.com,moni3837@yahoo.com

Abstract: **Location based spatial object selection and searching is emerging as an important search paradigm in indexing of multimedia database systems. The technique used is to map the objects as points into a two dimensional space which is indexed using a multidimensional data structure. Although several data structures have been proposed for location and spatial indexing, none of them is known to index points in both overlapping and non-overlapping regions in an optimum query retrieval time. This paper introduces the hybrid tree – a multidimensional data structure for indexing two dimensional location spaces. Unlike other multidimensional data structures, the hybrid tree cannot be classified as either a pure data partitioning (DP) index structure (e.g., R-tree, SS-tree, SRtree) or a pure space partitioning (SP) one (e.g., KDtree, hBtree);rather, it "combines" positive aspects of the two types of index structures a single data structure to achieve search performance is increased by a time complexity of *O(nlog(log n))* than either of the above techniques (hence, the name "hybrid"). Our experiments on "real" distributed large size spatial databases demonstrate that the fan out is independent of dimensionality and enables fast intra node search. It significantly outperforms purely k-d tree space partitioning and R-tree data partitioning-based index mechanisms as well as linear scan at all dimensionalities for large sized databases.**

Keywords: **Multimedia, multidimensional data, R-trees, k-d trees,fanout, spatial databases, indexing, data partitioning(DP), space partitioning(SP).**

1. Introduction

The role of multimedia and spatial databases is continuously increasing in many modern applications during last years. Mapping, urban planning, transportation planning, resource management, geomarketing, smart computing and robotics, archeology and environmental modeling are just some of these applications. The key characteristic that makes it a powerful tool is its ability to manipulate spatial data, apart from storing and representing them. The most basic form of such a manipulation is answering queries related to the spatial properties of data. Some typical spatial queries include selections with respect to a reference object (point location query; range query; nearest neighbor query) and joins between two spatial datasets (overlap or distance join). In this paper, the cost of a spatial query that combines join and nearest neighbor queries is studied.

This paper introduces hybrid tree which distinguishes itself from other multidimensional data structures. It is *neither a pure DP-based nor a pure SP based technique*. Bounding Region based techniques like R-trees tend to have low fanout and a high degree of overlap between bounding regions (BRs) at high dimensions. On the other hand, SP based techniques like k-d tree have fanout independent of dimensionality and no overlap between subspaces. The main contribution of this paper is the *"hybrid"* approach to multidimensional indexing: a technique that combines positive aspects of both the types of index structures to develop a single data structure.

- On one hand, like SP-based index structures, the hybrid tree: performs node splitting based on a single dimension and represents SP using kd-trees. This makes the fan out independent of dimensionality and enables fast intra node search.
- On the other hand, space partitions, like the Bounding Rectangles in DP-based techniques, are allowed to overlap whenever clean splits requires downward cascading splits, thus retaining the guaranteed utilization property.

The tree construction algorithms in the hybrid tree are geared towards providing optimal search performance. As desired, the hybrid tree allows search based on arbitrary location and regional functions. The regions can be specified by the user at query time for faster retrieval of neighboring nodes' or surrounding nodes' data in an optimized time.

The rest of the paper is organized as follows. In Section 2, we develop a classification of these data structures that allows us to compare them to the hybrid tree. Section 3 introduces the hybrid tree and is the main contribution of this paper. In Section 4, we present the performance results. Section 5 offers the final concluding remarks and future work.

2. Related Works

The increasing need of applications to be able to store objects in a database and index them based on their content has triggered a lot of research on multidimensional index structures. In this section, we develop a classification of multidimensional indexing techniques which allows us to compare the hybrid tree with the previous research in this area.

In multidimensional indexing trees, the overlapping of nodes will tend to degrade query performance, as one single point query may need to traverse multiple branches of the tree if the query point is in an overlapped area[7].In our hybrid tree indexing structure, we provide an optimal search performance by allowing the search based on arbitrary location and regional functions.

In Bhide et al,the LRU buffer replacement policy for databases,query performance is mainly affected by the time required to retrieve nodes touched by the query which do not reside in the buffer, as the CPU time required to retrieve and process buffer resident nodes is usually negligible[21].In hybrid tree structure, we solve the problem by treating the indexed subspaces as BRs in a DP-based data structure (which can overlap). We define a mapping the kd-tree based representation to an “array of BRs” representation. This allows us to directly apply the search, insertion and deletion algorithms used in DP-based data structures to the hybrid tree.

BDE(Binary Differential Evolution) algorithm selects the best feature subsets .The relativity of attributes is evaluated based on the idea of mutual information. The approach uses a population-based heuristics to evaluate the worth of features. This method was found very effective to improve the correct classification rate on some datasets, but was not comparitively efficient to extract the dataset from a relatively large database. This probelm can be solved by hybrid data structure which has a better time complexity and works faster than the other featureextraction models.

A strategy to perform a window query is first to decompose the corresponding window W into square subwindows, then the window query becomes the integration of the smaller subqueries over smaller subwindows. These subwindows are named maximal quadtree blocks.The subwindows are the blocks corresponding to the leaf nodes in the quadtree, which represent the window region within the image space[14].The limitation was that these types of quad-trees were not able to deal with overlapping regions. Hybrid trees on the other hand, uses bounding region based techniques that tend to have low fanout and a high degree of overlap between bounding regions (BRs) at high dimensions.

The features of multimedia data are useful for discriminating between multimedia objects.The SOM R-tree eliminates the empty nodes that causes unnecessary disk access and degrade the retrieval performance in case of poorly structured trees the disk access time is increased, thus leading to a poor performance[25].In r-kd tree structure, we reduce the time-complexity by using minimum bounding rectangles which automatically ignores the empty nodes.

3. Hybrid Tree:

First, we describe the “space partitioning strategy” in the hybrid tree i.e. how to partition the space into subspaces . First issue to be considered is the number of dimensions required to partition the entire region of space. As given in the Table1 we k-d partitioning is carried out resulting in the higher number of fanouts with the increasing dimensionality. the hybrid tree splits a node using a *single* dimension. 1-d split is the *only* way to guarantee that the fanout is totally independent of dimensionality. This is in sharp contrast with DP-based technique, which are at the other extreme: they use all the k dimensions to split, leading to a linear decrease in fanout with increase in dimensionality. Moreover on using the SP based indexing structure alone single dimension splits in the kd-tree necessitate costly cascading splits and causes creation of empty nodes. Due to the above reasons, kd-tree shows poor performance even in 4 dimensional location spaces. kd-trees cause cascading splits since it requires the node splits to be necessarily *clean* i.e. the split *must* divide the indexed space into two mutually disjoint partitions. We relax the above constraint in the hybrid tree:

- The indexed subspaces need *not* be mutually disjoint.
- The overlap is allowed only when trying to achieve an overlap-free would cause downward cascading splits and hence a possible violation of utilization constraints.

The splitting strategies of the various index structures are summarized in the Table 4.1. Since regular kd-trees can represent only overlap free splits, we need to modify the kd-tree in order to represent possibly overlapping splits. Each region of space is initially partitioned using the space partitioning technique into different partition of spaces . The space partitioning within each index node in a hybrid tree is represented using a kd-tree. Since regular kd-trees can represent only overlap free splits, we need to modify the kd-tree in order to represent possibly overlapping splits. Each internal node of the regular kd-tree represents a split by storing the split dimension and the split position. We add a second split position field to the kd-tree internal node ,i.e. a k-d tree single dimensional split is carried out on that particular region in order to spot the node’s exact neighbour from the k-d tree split. The first split position represents the right (higher side) boundary of the left (lower side) partition (denoted by *lsp* .or left side partition) while the second split position represents the left boundary of the right partition (denoted by *rsp* or right side partition). While *lsp=rsp* means non-overlapping partitions, *lsp>rsp* indicate overlapping partitions. The second change is in the algorithms for regular tree operations, namely, search, insertion and deletion. The tree operations in SP-based index structures are based on the assumption that the partitions are mutually disjoint. This is not true for the hybrid tree. We solve the problem by treating the indexed subspaces as BRs in a DP-based data structure (which can overlap). In other words, we define a *mapping* the kd-tree based representation to an “array of BRs” representation. This allows us to directly apply the search, insertion and deletion algorithms used in DP-based data structures to the hybrid tree. The mapping is defined recursively as follows: *Given any index node . of the hybrid tree and the BR .corresponding to it, we define the BRs corresponding to each child of N.* The BR of the root node of the hybrid tree is the entire data space. Given that, the above “mapping” can compute the BR of any hybrid tree node.

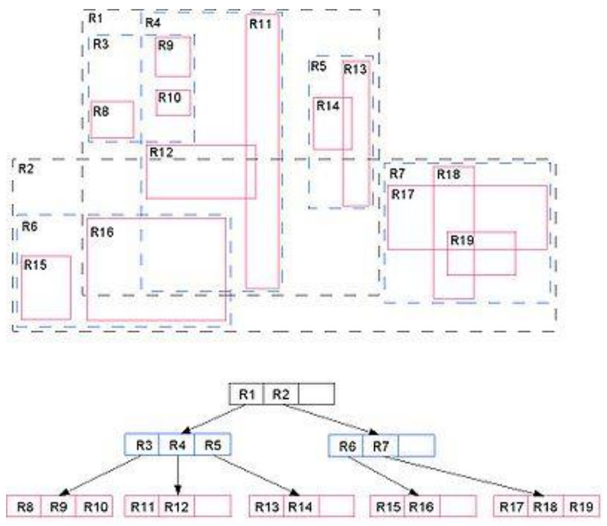
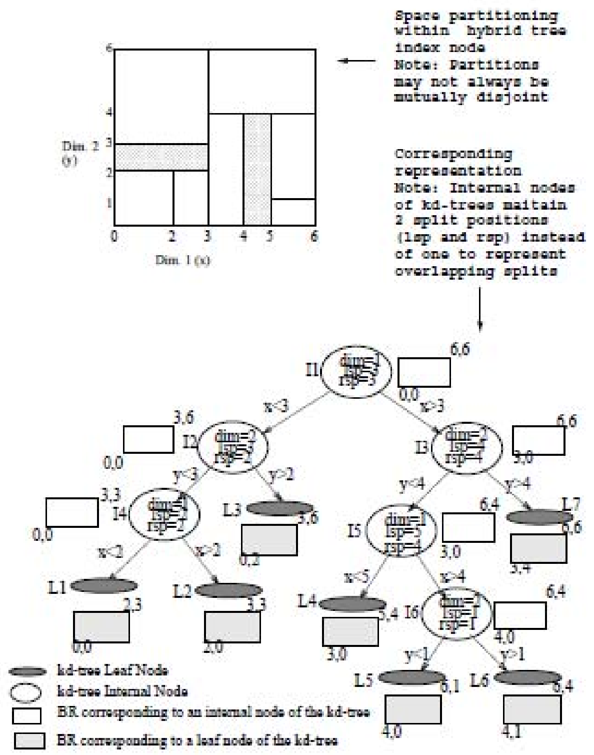


Figure3.1.DataPartitioning(BRs) of sample points.

Let N be an index node of the hybrid tree. Let Kn be the kdtree that represents the space partitioning within N and Rn be the BR of N . We define a BR associated with each node both internal as well as leaf nodes) of Kn. This defines the BRs of the children of N since the leaf nodes of Kn are the children of N. For example, the leaf nodes L1to L7are the children of the hybrid tree node N shown in the Figure 3.2. The BR associated with the root of KN is RN. Now given an internal node I of Kn and the corresponding BR RI, the BRs of the two children of are defined as follows. Let I={*dim,lsp,rsp*} where *dim,lsp*and *rsp* are the split dimension, left split position and right split position respectively. The BR of the left child of I is defined as RI∩(dim≤lsp)where, in the expression (dim≤lsp), dim denotes the variable that represents the value along dimension (for simplicity) and ∩,represents geometric intersection. Similarly, the BR of the right child of is defined as RI∩(dim≥rsp). For example, (0,0,0,6)is the BR for the hybrid tree node shown in Figure 3.2(BR is denoted as (xlo,ylo,xhi,yhi). The BR of I1 (the root) is (0,0,6,6) . The BRs of I2 and I3 are (0,0,6,6)∩(x≤3)=(0,0,3,6)and (0,0,6,6)∩(x>3)=;respectively. Similarly, the BR of L3, which, being a leaf of Kn, is a child of N, is obtained by BR (I2)∩ (y≥2) i.e. (0,0,3,6)∩(y≥2)=(0,2,3,6). The children of internal nodes with lsp>rsp have overlapping BRs (e.g., BRs of I4 and L3 (children of I2) overlap). Figure3.2 shows all the BRs – the shaded rectangles are the BRs of the children of the node while the white ones correspond to the internal nodes of Kn. Note that the above mapping is “logical”. The search/ insert/delete algorithm does not actually compute the “array of BRs” during tree traversal: rather it navigates the node using the kd-tree and computes the BR only when necessary . The kd-tree based navigation allows faster intranode search compared to array-based navigation. While searching for a correct lower level node using a kd-tree usually requires order log n comparisons (for a balanced kd-tree), searching in a array requires linear number of comparisons. Also, in a kd-tree representation, BRs share boundaries. In an array representation, the boundaries are checked redundantly while in a kd-tree, a boundary is checked only once.



Space partitioning within hybrid tree index node
Note: Partitions may not always be mutually disjoint
Corresponding representation
Note: Internal nodes of kd-trees maitain 2 split positions (lsp and rsp) instead of one to represent overlapping splits
Dim. 2 (y)
Dim. 1 (x)
I1 dim=1 lsp=3 rsp=3
I2 dim=2 lsp=3 rsp=2
I3 dim=2 lsp=4 rsp=4
I4 dim=1 lsp=2 rsp=2
I5 dim=1 lsp=5 rsp=4
I6 dim=2 lsp=1 rsp=1
x<3
x>3
y<3
y>2
y<4
y>4
x<2
x>2
x<5
x>4
y<1
y>1
L1
L2
L3
L4
L5
L6
L7
kd-tree Leaf Node
kd-tree Internal Node
BR corresponding to an internal node of the kd-tree
BR corresponding to a leaf node of the kd-tree

Figure 3.2. Mapping between each node and the corresponding BR. The shaded area represents overlap between BRs

The above algorithm can be implemented in a real time example by taking a map as a sample image and performing splitting and partitioning over it to obtain a hybrid tree structure as shown in figure 3.3.

The image is initially partitioned by taking a median point as an index and a kd-tree partitioning is performed to split the bounding box I1(root) into left and right children I2 and I3 respectively. Similarly the root values I2 and I3 is repeatedly partitioned to finally obtain leaf nodes after which there cannot be any more partitioning or splitting.

The shaded portion observed in the figure 3.3 denotes the overlapping regions of L1 and L3,L2 and L3 and L4 and L5.Incase of overlapping regions it makes the identification of locations easier by observing that certain points or places as is considered in the sample image are common to more than one region. In such cases,the overlapping regions are considered as separate bounding boxes and k-d tree partitioning is carried over that particular region to find its neighbouring places or places that surround that particular place which is considered as reference to perform k-d tree partitioning.

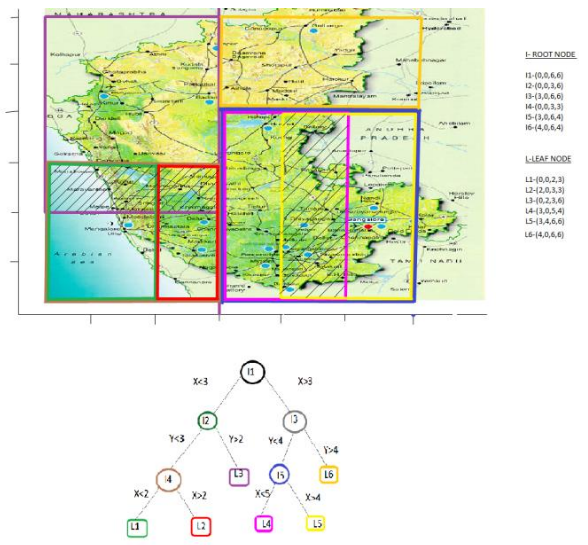


Figure 3.3. Hybrid Tree formation of a sample image

Once the partitioning and splitting is performed on the sample image, the regions are represented using a hybrid tree structure as shown in the figure 3.3.Root nodes and their respective leaf nodes are identified and the entries are made respectively in the hybrid tree structure .Thus it is seen clearly that the algorithm works well for any real time scenario, making the search performance effective when compared to the purely data partitioning and purely space partitioning techniques.

3.1.Data Node Splitting

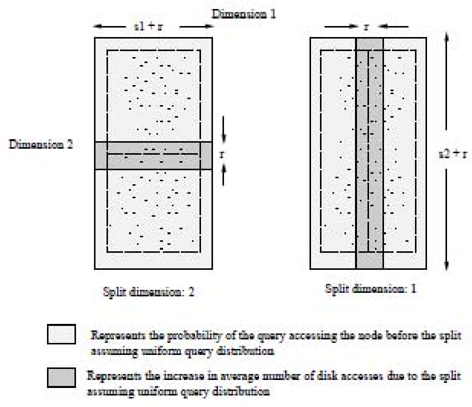
The choice of a split of a node consists of two parts: the choice of the split dimension and the split position(s). In this section, we discuss the choice of splits for data nodes in the hybrid tree.

Choice of split dimension: When a data node splits, it is replaced by two nodes. Assuming that the rest of the tree has not changed, the expected number of disk accesses per query (EDA) would increase due to the split. The hybrid tree chooses as the split dimension the one that minimizes the increase in EDA due to the split, thereby optimizing the expected search performance for future queries.

Let N be the data node being split. Let R be the k-dimensional BR associated with N. Let si be the extent of along the ith dimension,*i=[1,k]*. Consider a bounding box range query Q with each side of length . We assume that the location space is normalized (extent is from 0 to 1 along each dimension) and the queries are uniformly distributed in the data space. Let $P_{overlap}$ *(Q,R)* denote the probability that Q overlaps with R. To determine $P_{overlap}(Q,R)$, we move the center point of the query to each point of the data space marking the positions where the query rectangle intersects the BR. The resulting set of marked positions is called the Minkowski Sum which is the original BR having all sides extended by query side length r. Therefore,

$$P_{overlap}(Q,R)=(s1+r)(s2+r)..(sn+r) \quad (1)$$

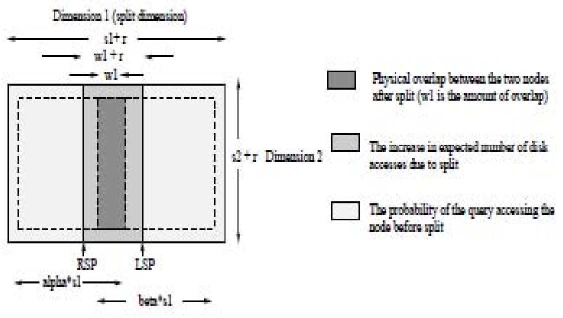
This is the probability that Q needs to access node N.



□ Represents the probability of the query accessing the node before the split assuming uniform query distribution

▨ Represents the increase in average number of disk accesses due to the split assuming uniform query distribution

Figure3.4 Choice of split dimension for data nodes.



■ Physical overlap between the two nodes after split (w1 is the amount of overlap)

▨ The increase in expected number of disk accesses due to split

□ The probability of the query accessing the node before split

Figure3.5. Index node splitting (with overlap)

Now let us consider the splitting of N and let j jbe the splitting dimension. Let N1and N2 be the nodes after the split and R1and R2 be the corresponding BRs.R1and R2 have the same extent as R along all dimensions except j i.e. *si,i=[1,k],i≠j*. Let αsj and βsj sjbe the extents of R1and 2along the jth dimension. Since the split is overlap-free, Rβ=1-α. The probabilities *Poverlap(Q;R1)*and *overlap(Q;R2)*are *(s+r) ..(αsj+r)..(sk+r)and (s1+r)* P*((1-α)sj+ r) …(sk+r)*respectively. Since R=R1UR2(where U is the geometric union) and Q is uniformly distributed,

$$P_{overlap}(Q;R1UR2)=P_{overlap}(Q;R1)overlap(Q;R2) \quad (2)$$

Thus, the probability $P_{overlap}(Q;R1) \cap overlap(Q;R2)$ that both N1and N2 are accessed is equal *to*

$$P_{overlap}(Q;R1) + P_{overlap}(Q;R2) - P_{overlap}(Q;R) \quad (3)$$

he volume of the dark shaded region in Figure3.2 is equal to

$$T \quad P_{overlap}(Q;R1) \cap overlap(Q;R2) \quad (4)$$

If Q does not overlap with R, there is no increase in number of disk accesses due to the split. If it does,

$$P_{overlap}(Q;R1) \cap overlap(Q;R2) \quad (5)$$

is the probability that the disk accesses increases by 1 due to the split. Thus, the conditional probability that overlaps with both R1and R2 given Q overlaps with R, i.e.

$$[P_{\text{overlap}}(Q,R1) \cap P_{\text{overlap}}(Q,R2)] / P_{\text{overlap}}(Q,R) \quad (6)$$

represents the increase in EDA due to the split. The increase in EDA if j is chosen as the split dimension evaluates out to be r/sj+r. Note that r/sj+r is minimum if j is chosen such that sj=maxki=1si, independent of the value of r. The hybrid tree always chooses the dimension along with the BR has the largest extent as the split dimension for splitting data nodes so as to minimize the increase in EDA due to the split.

An example of the choice of split dimension is shown in Figure3.3. Note that the optimality of the above choice is independent of the distribution of data. It is also independent of the choice of split position. Previous proposals regarding choice of splitting dimensions include arbitrary/round-robin [12] and maximum variance dimension [24]. The maximum variance dimension is chosen to make the choice insensitive to "outliers" [24]. Since the number of disk accesses to be made depends on the size of the subspaces indexed by data nodes and is independent of the actual distribution of data items within the subspace, presence or absence of "outliers" is inconsequential to the query performance. We performed experiments to compare our choice of maximum extent dimension as the splitting dimension with the maximum variance choice and is discussed is Section 5.

Choice of split position: The most common choice of the split position for data node splitting is the median [20, 16, 24]. The median choice, in general, distributes the data items equally among the two nodes (assuming unique median). The hybrid tree, however, chooses the split position as close to the middle as possible. This tends to produce more cubic BRs and hence ones with smaller surface areas. The smaller the surface area, the lower the probability that a range query overlaps with that BR, the lower the number of expected number of disk accesses.

3.2. Tree Operations

The hybrid tree, like other disk based index structures (e.g., B-tree, R-tree) is completely dynamic i.e. insertions, deletions and updates can occur interspersed with search queries without requiring any reorganization. The tree operations in the hybrid tree are similar to the R-trees i.e. indexed subspaces are treated as BRs but the kd-tree based organization is exploited to achieve faster intranode search. In addition to point and bounding-box queries (i.e. feature-based queries), the hybrid tree supports distance-based queries: both range and nearest neighbor queries. Unlike several index structures (e.g., distance-based index structures like SS-tree, M-tree), the hybrid tree, being a feature-based technique, can support queries with arbitrary distance measures.This is important advantage since the distance function can vary from query to query for the same feature or even between several iterations of the same query in a relevance feedback environment.

The insertion and deletion operations in the hybrid tree is also similar to that in R-trees. The insertion algorithm recursively picks the child node in which the new object should be inserted. The best candidate is the node that needs the minimum enlargement to accomodate the new object. Ties are broken based on the size of the BR.

4.Performance Evaluation:

We performed extensive experimentation to (1) evaluate the various design decisions made in the hybrid tree and (2) compare the hybrid tree with other competitive techniques.Finding the nearest point is an O(log N) operation in the case of randomly distributed points if N. Analyses of binary search trees has found that the worst case search time for an k-dimensional KD tree containing M nodes is given by the following equation.

$$t_{worst} = O(K \, . \, M^{1-1/k}) \quad (7)$$

The limitations of this structure is that these trees are not balanced and their structure depends on order of insertion of data.

In our proposal we combine the features of both R-tree and k-d tree. Since we incorporate the concept of the r-tree in kd-tree, the complexity for finding the nearest neighbour among the robots in a particular region is reduced by avoiding the traversal in the intersected regions of the bounding rectangles .Thus, range search problem of the kd tree is solved using the r-kd tree.

The complexity of the r kd tree is calculated as follows:

- Building a static r-kd-tree from n points takes O (n log ^{2} n) time if an O (n log n) sort is used to compute the median at each level. The complexity is O (n log n) if a linear median-finding algorithm is used.
- Inserting a new point into a balanced r-kd-tree takes O (log n) time.
- Removing a point from the balanced r-kd tree takes O (log n) time.
- Finally the query complexity is calculated as O(n log(log n)) by combining the two indexing structures.

Figures 4.1 compares the different techniques in terms of their scalability to very large databases. The hybrid tree in Figure4.1 significantly outperforms all other techniques by more than an order of magnitude for all database sizes. The hybrid tree shows a decreasing normalized cost with increase in database size indicating sublinear growth of the actual cost with database size. It compares the query performance of various techniques 2 for distance-based queries. Again, the hybrid tree outperforms the other techniques. From the experiments, we can conclude that the hybrid tree scales well to high dimensional feature spaces, large database sizes and efficiently supports arbitrary distance measures.

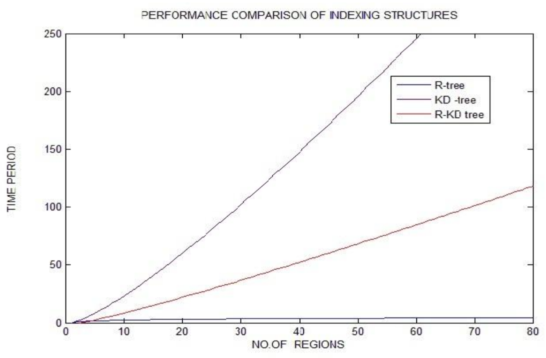


Figure4.1. Performance comparison of indexing structures

5.Conclusion

Location based similarity search is emerging as an important search paradigm in database systems. Efficient support of similarity search requires robust feature indexing techniques. In this paper, we introduce the hybrid tree - a multidimensional data structure for indexing high dimensional feature spaces. The hybrid tree combines positive aspects of bounding region based and space partitioning based data structures into a single data structure to achieve better scalability. It supports queries based on arbitrary distance functions. Our experiments show that the hybrid tree is scalable to high dimensional feature spaces and provides efficient support of distance based retrieval. The hybrid tree is a fully operational software and is currently being deployed for feature indexing in MARS .

6.Future Work

As part of future work, we intend to support new types of queries like approximate nearest neighbor queries efficiently using the hybrid tree. We also plan to explore techniques to support queries in interactive environments (e.g., relevance feedback) efficiently using the hybrid tree.There is also scope for structural improvements in the moving objects instead of bounding rectangles hich take the shape of the object in motion in location based search.

Index Structuree	**Number of dimensions to split**	**Number of (k-1)-d hyperplanes used to split**	**Fanout**	**Degree of overlap**	**Node utilization Guarantee**	**Time Complexity**
K- D tree	1	1	High(Independent of k)	None	No	O(log n)
R tree	k	2k	Low for large k (∞ 1/k)	High	Yes	O(n log n)
Hybrid tree	k	1 or 2	High(Independent of k)	Low	Yes	O(nlog(log n))

Table4.1.Splitting strategies for various index structures where k is the total number of dimensions

References

[1] S. Berchtold, C. Bohm, D. Keim, and H. P. Kriegel. A costmodel for nearest neighbor search in high dimensional data spaces. *PODS*, 1997.
[2] S. Berchtold, C. Bohm, and H. P. Kriegel. The pyramid technique: Towards breaking the curse of dimensionality. *Proc. of ACM SIGMOD*, 1998.
[3] S. Berchtold and D. A. Keim. Indexing high-dimensional spaces: Database support for next decade's application. *SIGMOD Tutorial*, 1998.
[4] S. Berchtold,D. A. Keim, and H. P. Kriegel. The x-tree: An index structure for high-dimensional data. *Proc. of VLDB*, 1996.
[5] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? *Proc. of ICDT*, 1999.
[6] T. Bozkaya andM. Ozsoyoglu. Distance-based indexing for high dimensional metric spaces. *Proc. of SIGMOD*, 1997.
[7] Chang-Tien Lu, Member, IEEE, Jing Dai, Student Member, IEEE, Ying Jin, and Janak Mathuria,GLIP: A Concurrency Control Protocol for Clipping Indexing *IEEE ,T Transactions on Knowledge and Data Engineering, VOL. 21, NO. 5, MAY 2009*
[8] T. Chiueh. Content-based image indexing. *Proc. of VLDB*, 1994.
[9] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. *Proc. of VLDB*,1997.
[10] Dongxiang Zhang, Yeow Meng Chee , Anirban Mondal, Anthony K. H. Tung, Masaru Kitsuregawa
Keyword Search in Spatial Databases: Towards Searching by Document, IEEE International Conference on Data Engineering
[11] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. ACM SIGMOD Conf., pp. 47–57.*, 1984.
[12] A. Henrich. The lsdh-tree: An access structure for feature vectors. *Proceedings of ICDE*, 1998.
[13] Y. Ishikawa, R. Subramanya, and C. Faloutsos. Mindreader: Querying databases through multiple examples. *Proc. of VLDB*,1998.
[14] N. Katayama and S. Satoh. The sr-tree: An index structure for high dimensional nearest neighbor queries. *Proc. of SIGMOD*, 1997.
[15] Kun-seok Oh, Yaokai Feng, Kunihiko Kaneko, Akifumi Makinouchi, SOM-Based R*-Tree for Similarity Retrieval, 2001 IEEE
[16] D. Lomet and B. Salzberg. The hb-tree: A multiattribute indexing mechanism with good guaraneed performance. *ACM Transactions on Database Systems*, 15(4), 1990.
[17] J. Nievergelt, H. Hinterberger, and K. Sevcik. The grid file: An adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems(TODS)*, 1984.
[18] M. Ortega, Y. Rui, K. Chakrabarti, S. Mehrotra, and T. Huang. Supporting similarity queries in mars. *Proc. of ACM Multimedia 1997*, 1997.
[19] M. Ortega, Y. Rui, K.Chakrabarti, S. Mehrotra, and T. Huang. Supporting ranked boolean similarity queries in mars. *Accepted for publication in IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 1998.
[20] J. T. Robinson. The k-d-b-tree: A search structure for large multidimensional dynamic indexes. In *Proc. ACM SIGMOD*, 1981.
[21] Scott T. Leutenegger Mario A. Lopez, The Effect of Buffering on the Performance of R-Trees[22] R. Weber, H. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high dimensional spaces. *Proc. of VLDB*, 1998.
[22] D. White and R. Jain. Similarity indexing with the ss-tree. *Proc. of ICDE*, 1995.
[23] D. White and R. Jain. Similarity indexing: Algorithms and performance.*Proc. of SPIE*, 1996.
[24] Yan Dong, Qingqing Zhang, Na Sun, Xingshi He Feature selection with discrete binary differential evolution, 2009 International Conference on Artificial Intelligence and Computational Intelligence