http://ansinet.com/itj

ITJ

ISSN 1812-5638

# INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 5 (6): 1038-1042, 2006
ISSN 1812-5638


# Building High Dimensional Indexing Structure Based on Cluster Information for Better Space Utilization

[1]Sanjay Garg and [2]Ramesh Chandra Jain
Department of Computer Engineering, A.D. Patel Institute of Technology,
New V.V. Nagar, Gujarat State, India
[2]Department of Computer Applications, S.A.T.I. (Engineering College), Vidisha,
Madhya Pardesh, India

**Abstract:** An index structure organizes a data set to support efficient queries and clustering is an unsupervised analysis technique for discovering interesting data distributions and patterns in underlying data sets. Many approaches have been proposed to index multidimensional data sets. Even though many approaches can insert data points dynamically, their performance depends on the insertion order. An attempt has been made to build a variation of R tree multidimensional indexing structure using cluster information termed as cluster-R tree. This approach splits a node on the basis of clustering rather than splitting a node in equal partitions like R-tree. Each interior node of cluster-R tree contains cluster of rectangles and exterior nodes contain clusters of data points. This approach outperforms cluster tree and some other classical high dimensional indexing techniques. Approach presented in this study is not sensitive to insertion order. The objective of this study is to investigate the space utilization feature Cluster-R tree structure that considerably improves the performance in indexing high dimensional data by incorporating the cluster information in the structures. This study presents space utilization aspect of proposed algorithm that is experimentally evaluated.

**Key words:** High dimensional databases, clustering, indexing, nn-search, similarity queries

## INTRODUCTION

Unlike classification and prediction, which analyze class-labeled data objects, clustering analyzes data objects without consulting a known class label. In general the class labels are not present in the training data simply because they are not known to begin with. Clustering can be used to generate such labels. The objects are clustered or grouped based on principle of maximizing the interclass similarity and minimizing the interclass similarity. So that clusters of objects are formed so that objects within a cluster have high similarity in comparison to one another, but are very dissimilar to objects in other clusters. Each cluster that is formed can be viewed as a class of objects, from which rules can be derived (Zait *et al*., 1997; Garg *et al*., 2005c).

An index structure organizes the whole data set to support efficient queries. Recently many applications require efficient access and manipulation of large-scale multidimensional datasets. For example many features extracted from image datasets are high-dimensional vectors. Also, in bioinformatics, gene expression data extracted from the DNA micro array images form large scaled multidimensional datasets. The high dimensions and enormous size of these datasets pose very challenging problems in indexing of the datasets for efficient querying. The design of indices to support high-dimensional data access has become an active research area.

Existing multi-dimensional tree like indexing approaches can be classified into two categories viz. Data partitioning approach and Space partitioning approach.

Space partitioning approaches divide a data space into disjoint subspaces. A hierarchy of subspaces can be generated through a divisive process. The K-D-B tree (Robinson, 1981) is one of the earliest multidimensional index structures based on space partitioning approach. The Pyramid tree and the Hybrid tree (Chakrabarti *et al*., 2000) are also fall in to category of space partitioning approach.

A data partitioning approach partitions a data set and builds a hierarchy of bounding In data partitioning, generally the index structures are having two kinds of nodes: internal and leaf nodes. The R tree (Guttman, 1984) and its variants R* Tree (Beckmann *et al*., 1990) etc. use a rectangle as the bounding shape.

**Corresponding Author:** Sanjay Garg, Computer Engineering Department, A.D. Patel Institute of Technology,
New V.V. Nagar, Gujarat State, India Tel: +91 2692 236856

The SS tree (White *et al*., 1996) is a similarity indexing strategy for high-dimensional data sets. Compared to the R tree the SS tree has higher fan-out because the hyper spheres for regions require half the storage space of the hyper rectangles. SS+ (Kurniawati *et al*., 1997) tree is an improvement in SS tree for the efficient similarity search.

The SR tree (Katayama *et al*., 1997) is a index structure which combines the bounding spheres and rectangles for the shapes of node regions to reduce the blank entries. SR tree takes the advantages of both rectangles and spheres and enhances the query performance remarkably. However, the storage required for the SR tree is larger compared to all other trees.

A cluster tree (Dantong *et al*., 2003) is a hierarchical representations of the cluster of a data set. A cluster tree organizes the date based on their different level of clustering information, from coarse to fine, providing an index structure of data. basic idea of this algorithm to decompose a cluster into subclusters to form an index structure. Node shapes and other features are opted from SS+tree indexing structure.

## BACKGROUND: THE R TREE

Here, a description of R-trees(Guttman, 1984; Gavrila, 1994) as the most important data structure for indexing rectangles is presented.

The R-tree is a height balanced tree-like data structure, similar to the B-tree. The main goal of an R-tree is to organize the spatial data in such a way that a search will visit as few spatial objects as possible before returning the result. The decision on what nodes to visit is made based on the evaluation of spatial predicates, so the tree must be able to hold some sort of spatial data on all nodes.

The R-tree is height-balanced tree in which index records in its leaf nodes contains pointers to data objects. The leaf nodes index record entries are of the form

$$(I, o_{id})$$

where *I* is the Minimum Bounding Rectangle (MBR) of the indexed spatial object and $o_{id}$ is a pointer to a database record that contains the actual object.

The non-leaf nodes contain index records of the form

$$(I, cp)$$

where I is the MBR of all the rectangles present in the entries of the lower level child node pointed to by cp. for a graphical representation of an R-tree.

**MBRs in R tree:** MBR is an n-dimensional Minimal Bounding Rectangle used in R trees, which is the minimal bounding n-dimensional rectangle, bounds its corresponding objects. The spatial objects can be approximated by the bounding rectangle such that the area of the bounding rectangle should be minimum and it should cover the spatial data object completely.

An R-tree based index is completely dynamic in the sense that it allows concurrent searches and update operations. Also, no periodic reorganization is required.

The most typical example of a search is the one where the user asks for all objects overlapping a certain area. Since the MBRs stored in the index entries are allowed to overlap, the R-tree cannot guarantee that only one search path needs to be traversed. The challenge is to dynamically build up bounding boxes and distribute them in a way that will minimize the number of paths needed for solving a search.

**Search in R tree:** Searching an R-tree is done generally by finding all index records whose MBRs overlap a search rectangle. The algorithm will start with the root and will follow recursively all entries in the current node whose MBRs overlap the search rectangle. At the end of each path, a leaf node is processed and the MBRs of all entries in that leaf are tested against the search pattern. The index records found to have an overlapping rectangle are stored as a list of result candidates.

**Insertion in R tree:** Inserting index records for new data tuples is similar to insertion in a B-tree in that new index records are added to the leaves, nodes that overflow are split and splits propagate up the tree.

Starting from the root, if an entry of a node contains the data point, the corresponding child of that node will be checked to see whether its children's MBR still contain the random point. If so, the child's children will be again checked recursively until the random point cannot go down the tree. Due to the overlapping between bounding rectangles, one random point might end up in several bounding rectangles. The most related bounding rectangle would be selected, based on the area.

**Deletion in R tree:** To delete an entry from an R-Tree, first the search for entry would be performed on the tree. On finding the entry in the tree, that entry would be removed from the tree and the changes will be propagated upwards in the tree up to the root. If the root node has only one child after the tree has been adjusted, make the child the new root.

**Updates and Other Operations in R tree:** If a data tuple is updated so that is covering rectangle is changed, its index record must be deleted, updated and then re-inserted, so that it will find its way to the right place in the tree.

## HYBRID APPROACH: CLUSTER-R TREE

Cluster tree (Dantong *et al*., 2003) is basically a divisive approach and an attempt has been made to generate an multidimensional index using agglomerative approach for incorporating cluster information with it. Cluster-R tree (Garg *et al*., 2005a, b) is a technique which incorporates cluster information in a index structure. The Cluster-R tree is a height balanced tree-like data structure, similar to the R tree. The main goal of an Cluster-R-tree is to organize the spatial data in such a way that a search will visit as few spatial objects as possible before returning the result. In Cluster-R tree along with other information the cluster information is also stored.

A Cluster-R tree is a hierarchical representation of the clusters of a data set. A Cluster- Tree organizes the data based on their different levels of clustering information, in which a parent cluster is divided in to sub clusters (child clusters), providing an index structure of data. Unlike to Cluster-Tree, Intermediate node of Cluster-R Tree is a cluster of midpoints of its child regions and a leaf node is a cluster of objects. This midpoint is a virtual object or medoid (Kaufman and Rousseeuw, 1990) representing the middle position of the rectangle. Each normal node in Cluster-R-Tree is defined as

Node: [Node_id, $\gamma$, (Entry1 ,Entry 2,…Entry r), IsLeaf] ($m_{node}$ <= $\gamma$ <= $M_{node}$),

Entry: ($Sc_i$, $BR_i$,$Area_i$,$Mid_i$)

Where Node_id is the node identifier, $\gamma$ is the number of the entries in the node and $m_{node}$ and $M_{node}$ define the minimum and maximum number of entries in the node. An entry is created for each sub cluster of the cluster which the current non leaf node represents. In entry $Entry_i$, $SC_i$ is a pointer to the ith subcluster, $BR_i$ is the bounding rectangle for the subcluster and $Area_i$ is the area covered by the bounding rectangle and $Mid_i$ represents middle point of the bounding rectangle. The bounding rectangle is represented by ($x_{min}$, $y_{min}$, $x_{max}$, $y_{max}$). IsLeaf indicates whether the node is a leaf node or an intermediate node depending on its value.

Each leaf node contains pointer to the data points and has the structure same as the node structure discussed above, but having IsLeaf value equals to True and $x_{min}$ equals to $x_{max}$ and $y_{min}$ equals to $y_{max}$. We define the level of Cluster-R-Tree as the length of the path from the root to this node, beginning with 0. It is desirable that the Cluster-R-Tree should be balanced so that the levels of the leaf nodes are equal.

```
insert (element);
begin
(L, K') choosleaf (element);
if L is not full, install (element);
  else splitnode-and-install (L, k' element);
end.
```

Fig. 1: Procedure insert

```
Procedure splinode-and-install (L, k', element);
begin
    L; = kmean-cluster (L, k'+1);
     balance-cluster (L', k);
     adjusttree (L');
     insert (element);
end.
```

Fig. 2: Procedure splitnode-and-install

The construction of ClusterR-Tree is very similar to that of R-Tree. The ClusterR-Tree is constructed as the insertion of the data points is being done.

**Insertion in cluster-R tree:** Inserting index records for new data tuples is similar to insertion in a R-tree in that new index records are added to the leaves, nodes that overflow are split and splits propagate up the tree.This procedure is explained in Fig. 1. Procedure *chooseleaf* (elemnt) chooses an appropriate leaf node where element can be inserted and returns pointer *L* to chosen leaf and number k' of sibling clusters. If room is available then install (element) procedure inserts it, otherwise call splitnode-and-install (L, k',element).

Incorporating clustering algorithm here modifies traditional Split Node algorithm. In R-tree the Splitting of node is not considering the data distribution of the data to be inserted. So presented approach uses the clustering algorithms for performing the split operation. Any of the distance based clustering algorithms can be used to perform the split and k-means algorithm is used here to carry out the splitting.

When the number of entries in the node exceeds the value k (capacity of a node i.e., maximum number of clusters/elements can be accommodated in a node), that node has to be split and the information should be propagated upwards to the root of the tree. The Procedure splitnode-and-install (L, k',element) (Fig. 2) finds pointer L' to parent of L and divides the k' groups in k'+1 groups and assigns them to k'+1 different nodes provided k' is less than k. If k and k' are equal then split will propagate toward root and new L'(pointer to the node which is being reclusterd) is to be returned. Detected clusters of L' are to be balanced in such a way that a

cluster should have maximum k members and minimum k/2 members. Procedure balance-cluster (L', k) performs This balancing task. Finally balanced cluster information is used for the node reassignments by procedure adjust tree (L') at node L' and element is to be inserted through a call of insert (element) again.

An index structure must handle the dynamic insertion of new data points, which may require that the underlying clusters be adjusted. Most clustering approaches cannot add new data points efficiently, which greatly limits the flexibility of the clustering approaches. However, statically organized structures are normally more optimized than dynamically organized ones and, consequent, support queries more efficiently.

This approach is not sensitive to the insertion order of the new data points and can accommodate a large number of new data points and new clusters. By inserting the new data point into fairly close node, the index structure can coarsely partition the new data points, which greatly benefits the clustering on the new data points later on.

## RESULTS

To weight up the space utilization feature of the ClusterR-Tree we have carried out the experiments on synthetic data sets by varying number of dimensions, number of data items and number maximum outdegree of a node i.e. capacity of a node.

Space utilization criteria considered is average density of a node in the tree which is calculated as (1), deemed that n is the actual number of entries in the tree(considering all interior and leaf nodes), k is the capacity of a node and p is total number of nodes created.

$$\text{Space-utilization} = n/kp \quad (1)$$

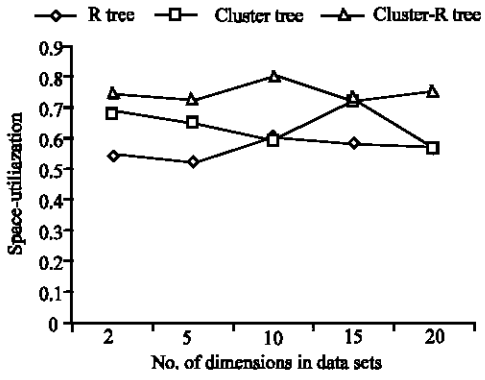


Fig. 3: Space utilization-varying number of dimensions, k = 8, size of datase = 5

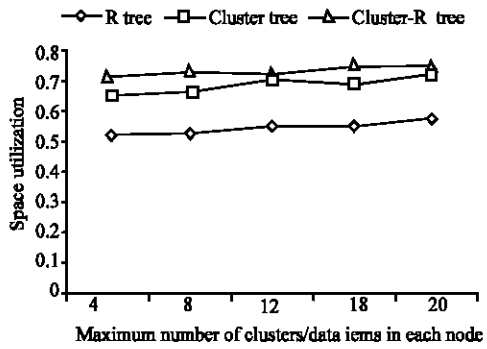


Fig. 4: Space utilization-varying k, size of dataset = 1,00,000, number of dimensions = 5

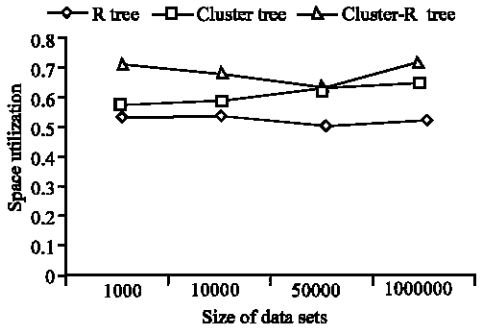


Fig. 5: Space utilization-varying size of dataset, k = 8, number of dimensions = 5

Figure 3-5 represents node density by varying number of dimensions, capacity of node (k) and size of database respectively while keeping other two parameters fixed (as shown in figure titles), which is experimentally estimated.

## DISCUSSION AND FUTURE EXTENSIONS

Space Utilization in Cluster-R-Tree are studied and In Most of the cases Cluster-R-Tree is over performing on other two indexing algorithms, of which it is a hybrid model. Some work is required to be done in the direction; which we could not infer through these experiments and a precise mathematical model to predict worst case complexity of clustering based indexing algorithms will also be very helpful. An indexing with higher space utilization always facilitate positively in the complexity of the search, such algorithms need to be explored, particularly clustering based indexing algorithms in order to perform similarity queries and NN-search.

## REFERENCES


Beckmann, N., H.P. Kriegel, R. Schneider and B. Seeger, 1990. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles in Proc. ACM-SIGMOD Int'l Conf. Management of Data, (Technical Report 94-13, University of British Columbia), pp: 322-331.

Chakrabarti, K. and S. Mehrotra, 2000. The hybrid tree: An index structure for high dimensional feature spaces. In Proc. 16th Int'l Conf. Data Eng., Feb., 2000, pp: 440-447.

Dantong Y. and A. Zang, 2003. Cluster Tree: Integration of cluster representation and nearest neighbor search for large datasets in high dimensions. IEEE Transactions on Knowledge and Data Eng., No. 5, September- October 2003, 15: 1316-1337.

Garg, S. and R.C. Jain, 2005a. Integration of R-Tree and Clustering technique for search efficient Indexing of high dimensional datasets. J. Computer Sci. (communicated and accepted).

Garg, S. and R.C. Jain, 2005b. On overlapping regions in cluster-R tree. In Proceedings of 5th Global Conference on FMS. Management of IT and Infrastructure for Flexibility and Competitiveness. 27-31 Dec. 2005, pp: 396-402.

Garg, S., A. Patel and R.C. Jain, 2005c. A comparative study of clustering algorithms for high-dimensional data. J. Eng. Technol. SPU Dec 2005, 18: 120-125.

Gavrila, D.M., 1994. R Tree Index Optimization in Advances in GIS Research. Waugh, T. and R. Healey (Eds.), Tayor and Frances.

Guttman, A., 1984. R-Trees: A dynamic index for geometric data. In Proceedings of ACM SIGMOID Int'l Conf. in Management of Data, pp: 73-84.

Katayama, N. and S. Saton, 1997. The SR tree: An index structure high dimensional nearest neighbor queries. In Proc. ACM SIGMOD Int'l Conf. Management of Data, pp: 369-380.

Kaufman, L. and P.J. Rousseeuw, 1990. Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley.

Kurniawati, R., J.S. Jin and J.A. Shepherd, 1997. The SS+ tree: An improved index structure for similarity searches in a high dimensional feature space. In Proc. Spie Conf. Storage and Retrival for Image and Video Databases, Feb. 1997, pp: 13-24.

Robinson, J.T., 1981. The KDB-tree: A search structure for large multi-dimensional dynamic indexes. In Proc. ACM SIGMOD Intl. Conf. on Management of Data, pp: 10-18.

White, D.A. and R. Jain, 1996. Similarity indexing with the SS-Tree in Proc. 12th Int'l Conf. Data Engg. Feb 1996, pp: 516-523.

Zait, M. and H. Messatfa, 1997. A comparative study of clustering methods. Future Generation Computer Systems. Nov. 1997, 13: 149-159.