DSR*-tree: Building and Retrieving

6X−8

Yaokai Feng, Masaaki Kubo, Akifumi Makinouchi

Graduate School of Information Science and Electrical Engineering,
Kyushu University

1 Introduction

In order to efficiently manage multi-dimensional data, one efficient index is very necessary. At present, R-tree family, especially its famous variant, R*-tree [1], is regarded as being among the most popular hierarchical structures for multidimensional indexing and is widely used in multimedia and spatial databases.

An R*-tree used for point objects is a hierarchy of nested d-dimensional MBRs (minimum bounding rectangles). Each non-leaf node of the R*-tree contains an array of entries, each of which consists of a pointer and an MBR. The pointer refers to one child node of this non-leaf node and the MBR is the minimum bounding rectangles of one child nodes referred to by the pointer. Each leaf node of the R-tree contains an array of entries, each of which consists of an object identifier and its corresponding point.

NN (Nearest Neighbor) search and range search are very popular in multimedia database (Find similar images) and spatial database (Find closest cities). According to our investigation, for a given database, the degree of clustering of objects in leaf nodes is a very important factor on the performance of the NN search and the range search. However, in R-trees, the objects are not well-clustered in the leaf nodes, especially when they are used to index skewed data in high-dimensional space. Some packing algorithms such as Hilbert Sort algorithm for R-trees have been proposed. However, these packing algorithms try to pack the same number of objects in each leaf node. It may not always lead to a good clustering. In order to improve the clustering degree, An attempt combining clustering technology and R-trees (called NSBR*-tree) is proposed by Y. Feng et al. [2]. However, its dynamic performance is not good. In this paper, a new structure called DSR*-tree (Dynamic SOM-based R*-tree) with a better dynamic performance is proposed.

2 NSBR*-tree

In NSBR*-tree, the clusters discovered by SOM are directly used to form "leaf nodes", which are contained in an array-like structure (called *ArrayPart*. All the MBRs of the clusters are used to build its *TreePart* (an R*-tree). Then, the TreePart and the *ArrayPart* are linked to form the NSBR*-tree.

However, the dynamic performance of the NSBR*-tree is not very good. After some object is inserted in or deleted from an NSBR*-tree, the *ArrayPart* and its link to the *TreePart* have to be rebuilt.

3 DSR*-tree

The dynamic performance of NSBR*-tree is not very good is because of its *ArrayPart*. In order to improve the dynamic performance, the DSR*-tree does not employ the ArrayPart of the NSBR*-tree any longer.

Building algorithm

Let m and M refer to the minimum bound and the maximum bound of the number of entries in each leaf nodes of R*-tree, respectively.

- step 1: By SOM, the clusters of objects are discovered.
- step 2: Repeatedly scan all the clusters until the small clusters whose cardinalities are less than m do not exist any longer. For each small cluster like these, merge it with its nearest neighboring clusters.
- step 3: Scan all the clusters. For each large cluster whose cardinality is larger than M, invoke the **SplitCluster algorithm** to split it.
- step 4: Calculate the MBR for each cluster and employ all the cluster MBRs to build an R*-tree, which is the first part (called *R*-Part*) of the DSR*-tree. The leaf node of the *R*-Part* is called *p-node*.
- step 5: The objects in each cluster are intended to be placed in one node (called cluster-node).
- step 6: All the cluster-nodes are linked with the corresponding entry in the corresponding *p-node* of the *R*-Part*. All the cluster-nodes form the *Cluster-Node-Part*.

Figure 1 shows an example of DSR*-tree.

SplitCluster algorithm

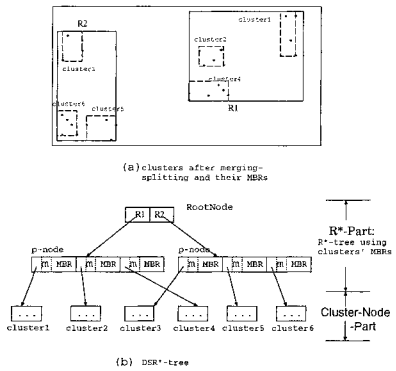

Figure 1: Example of DSR*-tree

A new operator with two integers is introduced as follows.

$$\left\{\frac{p}{q}\right\} = \text{the least integer that is not less than } \frac{p}{q}$$

Let the cluster that need to be split be *CLUSTER* and let its cardinality be C. Thus, *CLUSTER* should be split into $\{\frac{C}{M}\}$ groups, each of which has M objects (except the last one). The SplitCluster algorithm is described as follows.

1. Repeatedly perform the following operations for each axis.
   (a) Sort all the objects in *CLUSTER* by the coordinates in this axis.
   (b) All the objects in *CLUSTER* are ordered in $\{\frac{C}{M}\}$ consecutive groups of M objects. Note that the last group may contain fewer than M objects.
   (c) Calculate MBR for each group and calculate the sum of the volumes of all the MBRs. Let S refer to the sum of volumes.
2. Select the split axis and the packing method which have the smallest S.

4 Experiments

We used the following 12D-Image40000 database to test the behaviors of R*-tree, packed R-tree, NSBR*-tree and DSR*-tree. The execution time and the number of object distance calculations (which is regarded as the main cost in multimedia data searching) are tested with a hot cache.

12D-Image40000 40000 color images from H^2soft corporation, including pictures of landscapes, animals, buildings, people and plants. The image size is fixed at 128×128 pixels. Using a six-level two-dimensional wavelet transform, the dimensionality of image feature vectors is decreased to 12.

The experimental results of search performance are shown in Figure 2.

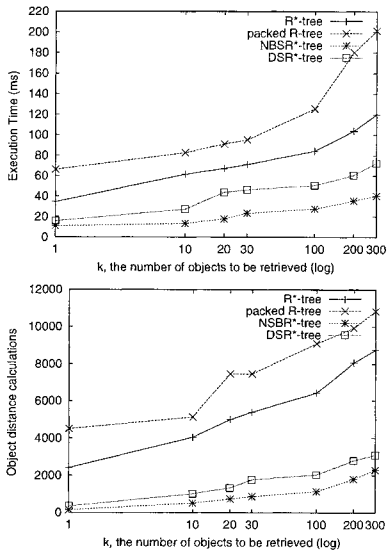

Figure 2: NN search performance comparison among R*-tree, packed R*-tree, NSBR*-tree and DSR*-tree.

5 Conclusion

The new index structure for point objects proposed in this paper is called DSR*-tree. The DSR*-tree combines the clustering technology and R*-tree in order to improve the clustering of the objects in the leaf nodes and to improve the search performance. The analysis and experimental results show that it has much better search performance than R*-tree and packed R*-tree. The search performance of the DSR*-tree is slightly worse than NSBR*-tree. However its dynamic performance is better than NSBR*-tree.

References

[1] N. Beckmann, H.P. Kriegel, R. Schneider, B. Seeger. "The *R**-tree: An Efficient and Robust Access Method for Points and Rectangles.". In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 322–331, May 1990.

[2] Y. Feng, M. Kubo et al. "NSBR*-tree: Building and Retrieving". In *Database Workshop (dbws2001)*, Hakodate, Japan, July 2001.