データベースシステム 125−49
（2001. 7. 18）

新提案NSBR*-tree: 構築と検索

FENG Yaokai*, 久保 正明 *, AGHBARI Zaher**, 牧之内 顕文 **

* 九州大学大学院システム情報科学府
** 九州大学大学院システム情報科学研究院

R-treeは多次元データにインデックスをつけるために使われている技術であり、空間、多次元データベースにおいて広く利用されている。しかし、オブジェクトは葉ノードにおいてうまくクラスタリングされていない。この問題はR-treeでの近傍探索時に大きく影響を与えるものである。本論文では全く新しいSOM-based R*-tree(NSBR*-tree)を提案する。この手法を 用いて葉ノードの重なりを減らし、オブジェクトをうまくクラスタリングさせることを試みる。実験結果ではこの新しいSOM-based R*-treeがすばらしい検索性能を持っていることを示す。

NSBR*-tree: Building and Retrieving

FENG Yaokai, KUBO Masaaki, AGHBARI Zaher, MAKINOUCHI Akifumi

Graduate School of Information Science and Electrical Engineering,
Kyushu University

Abstract

R-trees are a common indexing technique for multi-dimensional data and are widely used in spatial and multi-dimensional databases. Nearest neighbor search (called NN search) is very popular in multimedia database and spatial database. According to our investigation, for a given database, the degree of the leaf nodes clustering the objects is a great factor on the NN searching performance. For R-trees, the objects are not well-clustered by its leaf nodes. Some packing algorithms for R-trees have been proposed. However, in these packing algorithms, the distribution of objects in its leaf nodes may not reflect the actual situation of objects and can not lead to a good clustering. An attempt combining clustering technology and R-trees (called SOM-based R*-tree) is proposed by K. Oh and Y. Feng et al., which tries to decrease the number of objects in R-trees by building R-trees using the representative feature vectors of clusters instead of objects themselves. In the present paper, a new structure called NSBR*-tree is proposed. The experimental result shows that the NSBR*-tree has a much better searching performance.

1 Introduction

In the present paper, we will consider point access method (PAM). In light of the increasing number of computer applications that rely heavily on multimedia data, the database community has recently focused on the management and retrieval of multimedia data (e.g., documents, images, video, music score, etc,). However, the indexing of multimedia data is usually a PAM issue because multimedia data are usually discriminated using their feature vectors, which are mapped into points in multidimensional space.

R-trees are a common indexing technique for multi-dimensional data and are widely used in spatial and multi-dimensional databases. Nearest neighbor search (called NN search) is very popular in multimedia database and spatial database. According to our investigation, for a given database, the degree of the leaf nodes clustering the objects is a great factor on the NN searching performance. For R-trees, the overlap among the leaf nodes is serious and the objects are not well-clustered by its leaf nodes, especially when it is used to index very skewed distributed data. Three main reasons for this are as follows:

1. The clustering function of R-trees is not strong.

Of course, the objects have been clustered by the leaf nodes after one R-trees is built. However, R-trees are nondeterministic in allocating the entries onto the nodes i.e., different sequences of insertions may build up different trees. Data inserted during the early growth of the structure may have introduced directory rectangles, which is not suitable to guarantee a good searching performance in the current situation. In R-tree, a very local reorganization of the directory rectangles is performed only during a split. Obviously, this reorganization is rather poor. In R*-tree, in order to improve the reorganization of the directory rectangles, a new policy called forced reinsertion is introduced. If a node overflows, it is not split right away. Rather, p entries are removed from the node and reinserted into the tree. The parameter p may vary. Beckmann et al.[1] suggest it should be about 30% of the maximal number of entries per page. At the same time, R*-tree also improve the original splitting algorithm for trying to lessen the overlap between bucket regions at the same tree level. However, The forced reinsertion and the improvement of splitting algorithm can not completely solve the problem of reorganizing the directory rectangles, because the improved algorithms are invoked only when overflow occurs. Still, the objects can not be clustered very well by the leaf nodes of one R*-tree.

2. The minimum number of entries in each leaf node may degrade the clustering effect and may worsen the overlap in each level.

Of course, the minimum number of entries in each node can guarantee a minimum space utilizing. However, this limitation may degrade the clustering effect of R-tree leaf nodes. This is because in real application, there usually exist many small clusters each of which there are only a few (or only one) objects in and these clusters are distant to the others. If the cardinality of one small cluster is less than the minimum number limitation of entries in one leaf node, this cluster has to be merged with other distant cluster(s) to create one leaf node. In this case, the merging may lead to great enlargements of the rectangles of leaf nodes and may lead to a great enlargement of the overlap among leaf nodes.

3. The forced reinsertion do not always lead to a better clustering.

See Fig. 1. The objects outside the dotted circle, 30% of the present objects in this overflowed leaf node, will be forced reinserted into the tree. Obviously, this reinsertion does not always lead to a better clustering.

In this paper, we propose an improved R*-tree. Clustering technology is introduced to cluster the objects in advance. Clusters with different sizes based on the distribution of the given database are obtained and these clusters directly form the leaf nodes. No limitation exists on the number of entries in each leaf node. Thus, reinsertion and split of leaf nodes are unnecessary.

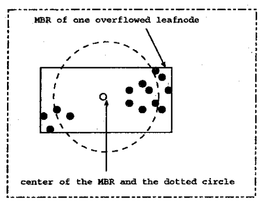

MBR of one overflowed leafnode
center of the MBR and the dotted circle

Figure 1: The choosing of reinserted objects in one leaf node

2 Related Work

2.1 Clustering Technology

Besides statistical techniques, artificial neural networks have proven as a successful tool in the cluster analysis. Zavrel [2] compared artificial neural networks with conventional techniques in the cluster analysis and found that neural networks generate significantly better results. Therefore, in our approach we employ SOM (Self-Organizing Map), a self-organizing neural network, for the discovery of clusters.

SOM is an unsupervised self-organizing neural network that is widely used to visualize and interpret large high-dimensional datasets [3, 4].

As a kind of neural network, SOM usually provides mapping from multi-dimensional points (feature vectors) onto points in a two-dimensional space. In this mapping, each feature vector is mapped onto one two-dimensional point with a code book vector (denoted CBV vector), which is nearest to the feature vector. In this way, many feature vectors nearly located each other may be mapped onto a same two-dimensional point. This means that the original multi-dimensional points are clustered.

2.2 R-trees

An R-trees used in a multimedia area is a hierarchy of nested d-dimensional MBRs (minimum bounding rectangles). MBR is a hyper-rectangle that minimally bounds the objects in the corresponding subtree. Each index 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 index node and the MBR is one of the child nodes referred to by the pointer. Each leaf node of the R-tree contains an array of entries, each of which consists of feature vector and its object identifier.

Based on a careful study of the original R-tree's behavior under different data distributions, Beckmann et al [1]. identified several weaknesses of the original algorithms in R-tree and propose R*-tree, an invariant of R-tree. In particular, they confirmed the observation of Roussopoulos and Leifker [5] that the insertion phase is critical for good search performance. It is also observed that the searching performance will be improved if some objects are reinserted after R-tree has been built, because reinsertion will re-organize the directory rectangles to make them much more suit the present data distribution. The design of the R*-tree therefore introduces a policy called forced reinsertion; If a node overflows, it is not split right away. Rather, p entries are removed from the node and reinserted into the tree. The parameter p may vary. Beckmann et al. suggest it should be about 30% of the maximal number of entries per page. The splitting algorithm is also improved to decrease the overlap in each level. However, all the improvement occurs only when overflowed. And, the searching performance will also be improved if some objects are reinserted after R*-tree has been built. This means that R*-tree can not also guarantee good clustering in each level.

2.3 Packed R-trees

Some packing algorithms on R-trees are proposed in [5, 6, 7]. by Roussopoulos, Kamel and Faloutsos, Scott T. Leutenegger et al. The general algorithms of the three packed algorithms are the same as follows:

1. Preprocess the data file so that all the objects (or rectangles) are ordered in consecutive groups of b objects, where b is the number of objects in each leaf node. Note that the last group may contain fewer than b objects (or rectangles).
2. Load all the groups of objects (or rectangles) into pages and output the (MBR, page-number) for each leaf level page into a temporary file. The page-numbers are used as the child pointers in the nodes of the next higher level.
3. Recursively pack these MBRs into nodes at the next level, proceeding upwards, until the root node is created.

The three algorithms differ only in the way how the objects (or rectangles) are ordered at each level. However, in all the three algorithms, each leaf node contains the same number of objects (or rectangles). Obviously, this can not reflect the actual distribution of objects. And, these algorithms can not lead to good clustering.

2.4 Existing SOM-based R*-tree

One SOM-based R*-tree has been proposed in [8]. The objects (feature vectors) are clustered in advance. The R*-tree is built using the CBV vectors as "objects", each of which corresponds to one cluster of objects. When k nearest neighbor search is performed, k nearest neighbor CBV vectors are obtained first, each of which corresponds to one cluster. Then, the k nearest neighbor objects are found in these k clusters. Obviously, the k actual nearest neighbor objects are not always located in the k "nearest" clusters. The unstrict searching result may be enough for some applications. However, it is not enough for many applications. The NSBR*-tree proposed in the present paper has overcome this problem. In our approach, the MBRs of all clusters obtained by clustering are employed as "objects" to build the R*-tree and all the objects in each of these clusters are contained in an array. All the searching algorithms on R*-tree can be used with a small modification and the searching result is strict.

3 NSBR*-tree

3.1 Introduction

In the NSBR*-tree proposed in this paper, the clusters obtained by SOM-based clustering are directly used to form "leaf nodes". No limitation exists on the number of entries in each leaf node. Thus, no overflow (neither reinsertion nor splitting) occurs in the "leaf node" level. That is, the distribution of "leaf nodes" can well reflect the actual distribution of objects. An array is introduced to contain all the objects of every cluster obtained by clustering.

The NSBR*-tree is shown in Fig. 2

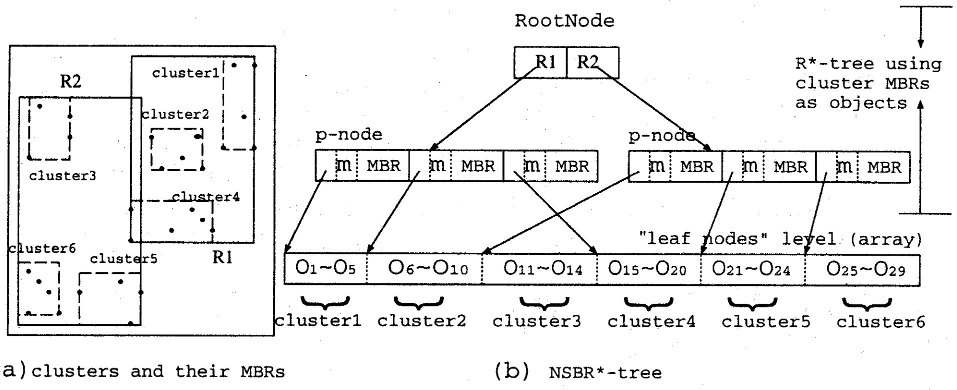


Figure 2: NSBR*-tree

Figure 2 (a) is the clusters discovered by SOM clustering; (b) is the corresponding NSBR*-tree. We think the clusters in the array may be regarded as "leaf nodes" of the NSBR*-tree, which are different from the other nodes. We call the parent nodes of the "leaf nodes" p-nodes. In p-node level, m refers to the number of objects in corresponding cluster and MBR refers to that of each cluster.

One problem occurs when the main memory does not have enough space for the array containing all the feature vectors since the array is loaded at once into the memory for searching. According to our investigations, for 100,000 12-dimensional feature vectors with floating values, 10M Bytes memory is enough. This is not a big problem for the present computers. Of course, for the applications using very large databases, the whole or part of the array can be held in the second storage. We think the NSBR*-tree can be applied well in applications with middle scale database or with lower scale database.

3.2 Building

Building algorithm of SOM-based R*-tree is as follows:

- **Step1:** using SOM technology, cluster all the objects;
- **Step2:** calculate the MBR of each cluster;
- **Step3:** build R*-tree using the MBRs obtained in Step2 as "objects" and the insertion algorithm of the original R*-tree.
- **Step4:** form an array containing all clusters;
- **Step5:** link the R*-tree obtained in Step3 and the array obtained in Step4;

3.3 Searching

Generally speaking, all the search algorithms on R*-tree can be simply adapted to NSBR*-tree. Here, the well-known k-NN search algorithm proposed by N. Roussopoulos in [9] is taken as an example.

The adopted k-NN search algorithm is shown in Fig. 3

```
Main procedure:: k-NN(QueryObject, k, IndexTree)
/* k-nearest neighbor search of QueryObject in IndexTree */
(1) NearList=new List(k) /* for k candidates of nearest neighbors */
(2) Distance of each member in NearList ← ∞     /* initialize NearList */
(3) k-NearestTraversal(QueryObject,k,NearList,Index.RootNode)
(4) report all objects in NearList as the final query result
END
Subprocedure:: k-NearestTraversal(QueryObject,k,NearList,Node)
1 If Node is a p-node Then
2     ActiveBranchList ← all entries in Node
3     Sort_by_distance(ActiveBranchList)
4     last=pruneActiveBranchList;
     /* last refers to the number of child_nodes left after pruning */
5     For i:=1 to last /* for each child_node left in ActiveBranchList;
6         if Dist(QueryObject, child_node_i)≤NearList.MaxDist Then
7           For each object of the corresponding cluster in array
8               If Dist(QueryObject,object)<NearList.MaxDist Then
9                   insert(NearList, Dist(QueryObject,object), Object)
10 Else /* if Node is not a p-node */
11    ActiveBranchList ← child_nodes in Node
12    Sort_by_distance(ActiveBranchList)
13    last=pruneActiveBranchList;
      /* last refers to the number of child_nodes left after pruned */
14    For i:=1 to last /* for each child_node left in ActiveBranchList */
15        if Dist(QueryObject, child_node_i)≤NearList.MaxDist Then
16            k-NearestTraversal(QueryObject,k,NearList,child_node_i)
14        else
15            exit For-loop
END
```

Figure 3: **adapted k-NN search algorithm for NSBR*-tree**

The main idea of this algorithm is the same as that of the k-NN search algorithm proposed by N. Roussopoulos [9]. The only difference is that, in the NSBR*-tree, when the tree is traversed recursively to some entry of some p-node, the corresponding cluster (not child node like R*-tree) in the array must be checked.

4 Experimental Results

We used a real color image database to examine the cost of building NSBR*-tree and the behavior of the k-NN search algorithm on it. Also, an R*-tree is built by inserting the objects one by one and the k-NN searching performance on the R*-tree is also tested. And the comparisons of the time cost for building, the index size and the k-NN searching performance are made between the R*-tree and the NSBR*-tree.

Color images from H^2soft corporation and Stanford University, including pictures of landscapes, animals, buildings, people and plants, are used in our experiment. The image size is fixed at 128×128 pixels. Haar wavelets, a kind of wavelet transform, are employed to compute their feature vectors because Haar wavelets are very fast and have been found to perform well in practice [10]. Using six-level two-dimensional wavelet transform, the length of image feature vectors decreases to 12.

The query objects are determined as follows. We chose 100 objects randomly from the database as the query objects. The index trees were built using the other objects. The cardinality of the database refers to the number of objects used to build the index trees.

Besides the time cost, we also tested the number of object distance calculations, which is regarded the main cost in multimedia data searching. All experiments were performed on a COMPAQ DESKPRO i386 (OS:FreeBSD 3.4-STABLE) having 128 MBytes of memory. The cardinality of database used in our experiment is 10000.

1. Comparison of index building

Table 1: Comparison of index building

items	R*-tree	NSBR*-tree
TimeCost of building index	12.4237(sec.)	7.0542(sec.)
height	4	4
index size	3.3MB	2.4MB

In Tab. 1, the timecost is calculated from the time within which all the necessary data for building index are loaded into memory. Thus, it does not contain the I/O operations. This is because the necessary data for building R*-tree and for building NSBR*-tree are the same, which results in I/O operations are the same. The index size for NSBR*-tree contains the size of the array.

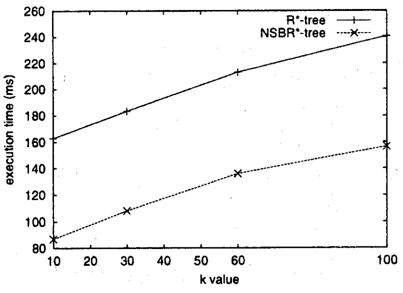
2. Comparison of k-NN Searching Performance

   The searching performance on the original R*-tree and NSBR*-tree are tested with different k values.

   Figure 4 shows the comparison of k-NN searching performance.

5 Conclusion

In the present paper, we combine the cluster technology and R*-tree in order to lessen the overlap among the leaf nodes and to well cluster the objects by the leaf nodes, which



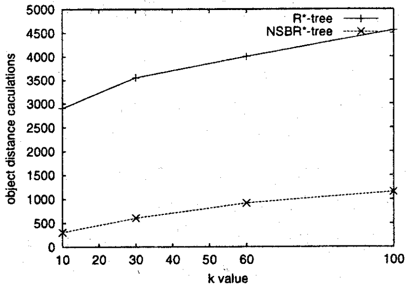


Figure 4: comparison on execution time and object distance calculations

are important issues to the searching performance. Our approach, a new index structure for point objects (called NSBR*-tree) has much better searching performance than the original R*-tree according to our experiments. The NSBR*-tree is described in detail, including its structure, its building and its searching algorithm.

References

[1] Beckmann N., Kriegel H.P., Schneider R., Seeger B. "The *R**-tree: An Efficient and Robust Access Method for Points and Rectangles.". In *ACM SIGMOD, May 1990.*

[2] J. Zavrel. *"Neural Information Retrieval"*. PhD thesis, University of Amsterdam, 1995.

[3] A. Rauber. "LabelSOM: On the Labeling of Self-Organizing Maps". In *Proc. of IJCNN'99*, Washington DC, July, 1999.

[4] T. Kohonen. "Self-Organization of Very Large Document Collections: State of the Art". In *Proc. of ICNN98*, volume 1, pages p.65–74, London, UK, 1998. Springer.

[5] Roussopoulos N. and Leifker D. "Direct Spatial Serach on Pictorial Databases Using Packed R-trees". In *Proceedings of the ACM SIGMOD international Conference on Management of Data, p.17-31, 1985.*

[6] Kamel I., Faloutsos C. "On Packing R-trees". In *Proc. 2nd International Conference on Information and Knowledge Management, p. 490-499, Arlington, VA, November 1993.*

[7] Scott T. Leutenegger et al. "STR: A Simple and Efficient Algorithm for R-tree Packing". In *Proc. of the 13rd International Conference on Data Engineer, p. 497-506 Birmingham U.K. 1997.*

[8] Kun-seok Oh, Yaokai Feng, Kunihiko Kaneko and Akifumi Makinouchi. "SOM-Based R*-Tree for Similarity Retrieval". In *DASFAA 2001, Hongkong, China, May, 2001.*

[9] Nick Roussopoulos, Stephen Kelley, Frederic Vincent. "Nearest Neighbor Queries". In *Proc. of ACM SIGMOD Intl. Conf. on Management of Data, San Jose, CA, p. 71-79, June, 1995.*

[10] Jacobs C.E., Finkelstein A., Salesin D.H. "Fast Multiresolution Image Querying". In *Proc. SIGGRAPH95, Los Angeles, California, p.6-11, 1995.*