A Proposal of Introducing Clustering Technology to R^*-tree

Feng, Yaokai
Department of Intelligent Systems, Graduate School of Information Science and Electrical Engineering, Kyushu University : Graduate Student

Makinouchi, Akifumi
Department of Intelligent Systems, Faculty of Information Science and Electrical Engineering, Kyushu University

https://doi.org/10.15017/1525448

出版情報：九州大学大学院システム情報科学紀要. 7 (2), pp.81-86, 2002-09-26. 九州大学大学院システム情報科学研究院
バージョン：
権利関係：

九州大学大学院
システム情報科学紀要
第7巻 第2号 平成14年9月

Research Reports on Information Science and
Electrical Engineering of Kyushu University
Vol.7, No.2, September 2002

A Proposal of Introducing Clustering Technology to R*-tree

Yaokai FENG*, Akifumi MAKINOUCHI**

(Received June 14, 2002)

Abstract: As a famous member of R-tree family, R*-tree is widely used in multimedia databases and spatial databases, in which NN (Nearest Neighbor) search is very popular. Based on the observation that the objects are not well-clustered in R*-tree leaf nodes, this paper proposes an approach to improve NN search performance of R*-tree by introducing clustering technology to R*-tree. The experimental result indicates that our improved R*-tree has much better NN search performance than the original R*-tree.

Keywords: Multidimensional index, Nearest neighbor search, Clustering, SOM

1. Introduction

Nearest neighbor (NN) search aims at finding the nearest objects to a given query point. In GIS, it can be used to find the nearest neighbors to a given location. In multi-media applications, it can be used to perform similarity search (content-based retrieval), which is very popular and can respond to the so-called "like this" query.

In order to efficiently manage multi-dimensional data, one efficient index is very necessary. At present, R-tree family, including R*-tree[1], X-tree[2], SR-tree[3] and so on, is still regarded as being among the most popular hierarchical structures for multidimensional indexing. However, according to our investigations, the objects are often not well-clustered in R*-tree leaf nodes, which is a very important factor on NN search performance. In this study, we introduce the clustering technology to indexing of multidimensional databases.

The rest of this paper is organized as follows. Section 2. Our observations on R*-tree are introduced in Section 2. Our approach is described in Section 4. Experimental results are presented in Section 5. Conclusions are drawn in Section 6.

2. Our Observations on R*-tree

In this section, after R*-tree is introduced briefly, we will explain the terms of bad clustering and good clustering, and the effect of clustering degree on NN search. And then, our observations on R*-tree are presented.

* Department of Intelligent Systems, Graduate Student
** Department of Intelligent Systems

2.1 R*-tree

An R*-tree 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 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 the corresponding point (for point objects) or the MBR of the corresponding object (for extended objects).

Figure 1 shows an example of R-tree.

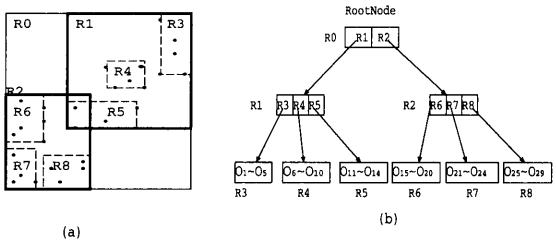

Fig.1 An example of R*-tree (a) data space (b) R-tree built from the data in (a).

2.2 Bad Clustering and Good Clustering

The distribution of the leaf nodes in **Fig.2 (a)** reflects that of the objects, whereas **Fig.2 (b)** is opposite. Thus, the clustering degree of objects in the leaf nodes in **Fig.2 (a)** is thought better than that in **Fig.2 (b)**. Certainly, the number of entries in each leaf node should meet the limitation on the

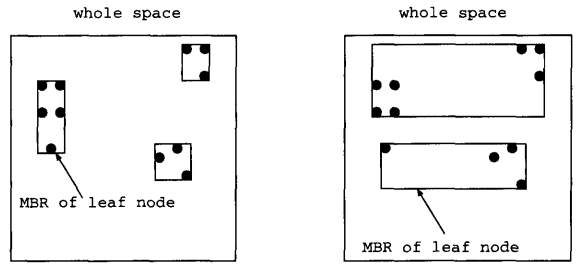

Fig.2 Visualized clustering degree.

fanout of R*-tree leaf nodes.

2.3 Effect of clustering degree on NN search

Generally speaking, the better the objects are clustered in the leaf nodes, the smaller the average size of the MBRs of the leaf nodes. This means that the better the objects are clustered in the leaf nodes, the farther the average distance of each leaf node from the query point (see **Fig.3**). In **Fig.3(a)**, the low clustering degree results in a smaller MINDIST of leaf node A than that of leaf node A' in **Fig.3(b)**.

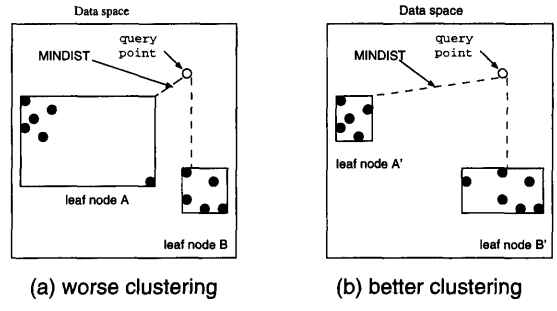

Fig.3 Effect of clustering degree on MINDIST.

There exist some Incremental NN (INN) search algorithms and the INN search algorithm[4] is regarded as the optimal NN search algorithm because of the minimum number of node accesses[5], in which a priority queue is used to contain nodes and objects that are still not dealt with. The members in the priority queue are sorted by their distances from the query point. Since a bad clustering of objects in leaf nodes tends to result in a short average distance of nodes from the query point, the nodes tend to be located before the objects in the priority queue. Thus, the nodes that have to be accessed tends to increase as the clustering degree degrades. This means that the clustering degree of the objects in leaf nodes also greatly affects performance of the INN search algorithm.

2.4 Weakness of R*-tree

R*-tree is well known as a common indexing technique for multi-dimensional data. However, the objects are not well-clustered in its leaf nodes, especially, when it is used to index objects with skewed distribution. There are the following main reasons:

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

Although the objects can be regarded as that they have been clustered in the leaf nodes after an R*-tree has been built, R*-tree is 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 search performance in the current situation.

Figure 4(a) shows an example of inserting one new object (the white point) in R*-tree. The insertion algorithm chooses one from the existing leaf nodes to contain the new object. Note that, the distribution of the existing leaf nodes is determined by the objects that have been inserted before and are not always suitable to the new object distribution after the new one is inserted. In other words, one insertion affects only one existing leaf node. Obviously, a better clustering of the objects in leaf nodes can be obtained if the leaf nodes are re-distributed as **Fig.4(b)** after this insertion, in which several leaf nodes are affected by this insertion.

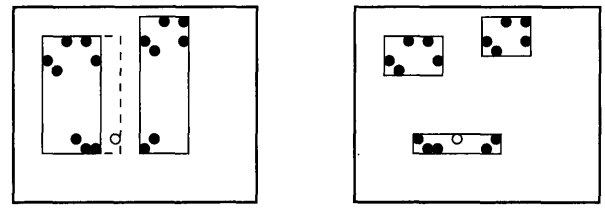

Fig.4 Insertion algorithm of R*-tree does not always lead to a good clustering.

2. The forced reinsertion does not always lead to a good clustering.

Reinsertion is an important feature of R*-tree. However, it can not guarantee a good clustering.

Figure 5(a) shows the situation of one reinsertion in R*-tree. The objects outside the dotted

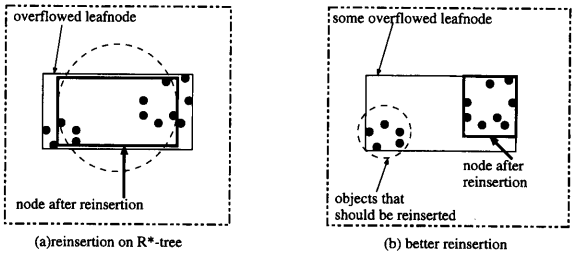

Fig.5 Forced reinsertion does not always lead to a good clustering.

circle will be reinserted in the R*-tree. Obviously, if the reinserted objects were chosen as **Fig.5(b)** (the objects in the dotted circle are reinserted), the node after this reinsertion is smaller and a better object clustering in the leaf nodes can be obtained.

3. Clustering Technology

There exist many clustering algorithms for discovering the cluster distribution of an object set. Besides statistical techniques such as CURE[6] and BIRCH[7], artificial neural networks (e.g., Self-Organizing Map) also have proven as successful tools in cluster analysis and have better performance (not worse at least) than the statistical techniques[8]. In this study, we use SOM (Self-Organizing Map) neural network, an unsupervised self-organizing neural network that is widely used to visualize and interpret large high-dimensional datasets[9),10].

As a kind of neural network, SOM provides mapping from a n-dimensional space where objects are distributed onto a two-dimensional map layer where many points (called map-points) exist (see **Fig.6**). Every object is mapped onto one map-point. Each of map-points has a n-dimensional vector (called *Codebook* vector) and a pair of coordinate values to indicate its position in the map layer. SOM is thought as *topological mapping*, which means that neighboring objects are mapped onto the same or the neighboring map-point(s). The objects that are mapped onto the same map-point are regarded as being in one cluster. In this way, the cluster distribution of the objects is discovered.

The process of SOM discovering cluster distribution can be divided into two phases of learning and clustering. In the learning phase, every *Codebook* vector of the map-points is randomly initiated. The *Codebook* vectors are adjusted as all the objects are input one by one. The learning phase is repeated many times to find more accurate *Codebook* vectors for the map-points according to the object

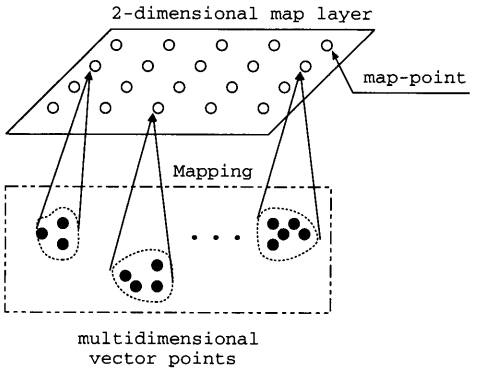

Fig.6 Clustering function of SOM.

distribution. In the clustering phase, each of objects is mapped onto one of the map-points whose *Codebook* vector is the closest one to this object.

The number of map-points in the map layer, *MapSize*, must be decided in advance. If this number is too large, many clusters may be discovered and there may be many very small clusters. On the other hand, if this number is too small, some very big clusters may occur. The issue of choosing *MapSize* will be discussed in Section 5.2.

4. Our Proposal

In the work[11], SOM is also introduced to R*-tree. However, it is for reducing the size of R*-tree and the query result of R*-tree is some clusters, not objects. In order to improve the clustering degree of the objects in leaf nodes, we have proposed an approach in the work[12]. In that approach, all the clusters of objects discovered by some clustering algorithm are packed in an array-like structure. It is for static databases only since an array is used.

Our proposal in this paper consists of two parts, Part 1 and Part 2. The Part 2 is formed from the clusters discovered by some clustering technology and Part 2 is an R*-tree built from all MBRs of the discovered clusters. **Figure 7** shows an example of our proposal.

Since there exist one maximum bound and one minimum bound on the number of entries in R*-tree nodes, preprocessing is necessary if some clusters are too small or too big after the cluster distribution is discovered.

4.1 Data Preprocessing

1. Clustering. The cluster distribution is discovered by SOM and MBR of each cluster is calculated.
2. Reorganizing of clusters.

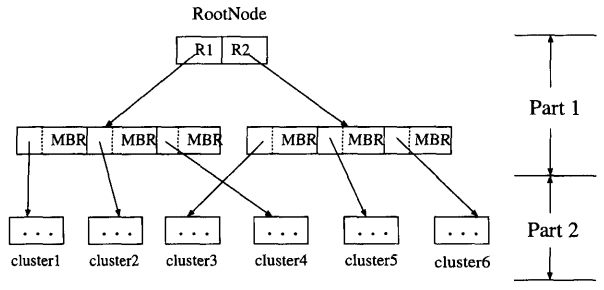

Fig.7 Our proposal.

Let m and M refer to the minimum bound and the maximum bound on the number of entries in each leaf nodes, respectively. the number of M determines node size and it is given by user. m should be about 40% of M according to the observation by N. Beckmann and et al.[1]. The clusters whose cardinalities are less than m are called *too-small-clusters*; The clusters whose cardinalities are greater than M are called *too-big-clusters*;

(a) Repeatedly scan all the clusters until no *too-small-clusters* exist any longer. For each *too-small-cluster*, invoke **MergingAlgorithm**.

(b) Repeatedly scan all the clusters until no *too-big-clusters* exist any longer. For each *too-big-cluster*, invoke **SplittingAlgorithm**.

MergingAlgorithm

Each *too-small-cluster* should be merged with some other cluster. MergingAlgorithm aims at finding which cluster it should be merged with.

Among all the clusters, MergingAlgorithm chooses the cluster whose MBR needs least volume enlargement to include this *too-small-cluster*. Resolve ties by choosing the cluster with the smallest MBR.

Note that, after merging, the cluster distribution should be updated and the MBR of the relevant cluster should also be re-computed.

SplittingAlgorithm

Let the cardinality of the *too-big-cluster* be C. Thus, this *too-big-cluster* should be split into $\lceil \frac{C}{M} \rceil$ groups, each of which has M objects (except the last one). SplittingAlgorithm is described as follows.

choosing splitting axis.

Repeatedly perform the following operations for each axis.

1. Sort all the objects in the *too-big-cluster* by the coordinates in this axis.
2. All the objects in the *too-big-cluster* are ordered in $\lceil \frac{C}{M} \rceil$ consecutive groups of M objects. Note that the last group may contain fewer than M objects.
3. Calculate MBR for each group and calculate the volume sum of all the MBRs. Let S refer to the sum of volumes.

Choose the split axis with the smallest S.

In the split axis, all the objects in the *too-big-cluster* are divided into $\lceil \frac{C}{M} \rceil$ consecutive groups of M objects. Again, the last group may contain fewer than M objects.

Note that, the issue may occurs that the last group of this splitting is *too-small-cluster*. That is, $t < m$, where t refers the number of objects in the last group. In this case, the last two groups will be divided equally. That means each of the last two groups has $(M + t)/2$ objects. Since $t < m$ and $2m < M$ (see the work[1]), $M > (M + t)/2 > m$ can be guaranteed. That is, the number of entries in each of the last two groups is in the range of $[m, M]$.

4.2 Index Building

Load all the clusters into pages, each of which forms one leaf node of Part 2. Output the (MBR, page-number) for each leaf level page into a temporary file, *TmpFile*. The page-numbers are used as the child pointers in the nodes of the next higher level.

4.3 Insertion, Deletion and Search

Our proposal is an improved R*-tree and it has the same structure as R*-tree. Thus, the insertion algorithm, deletion algorithm and any search algorithm on R*-tree can also be used in our proposal. However, if the database is updated to a great extent after the index is built then the benefit of utilizing clustering technology may become very weak. In this case, performance of our proposal tends to that of R*-tree and we think it is better to build the index again. Because search performance is paid close attention to for the relatively static databases, so we think that rebuilding the index is not a big problem.

5. Experiments

5.1 Databases and Environment

We used the databases of 40000 12-dimensional image data to test NN (nearest neighbor) search performance of our proposal.

12D-Image40000 40000 color images from H^2soft corporation[13], including pictures of landscapes, animals, buildings, people and plants. The image size is fixed at 128×128 pixels. In order to compute their feature vectors and to decrease their dimensionality, Haar wavelets (a kind of wavelet transform) are employed. Haar wavelets are very fast and have been found to perform well in practice[14]. Using a six-level two-dimensional wavelet transform, the dimensionality of image feature vectors is decreased to 12. This means that the 12D-Image40000 database consists of 40000 12-dimensional feature vectors.

All experiments were performed on an EPSON DIRECT PC having 128 MBytes of memory and FreeBSD 4.3 OS. We used the INN search algorithm[4], which is regarded as the optimal NN search algorithm[5].

The query objects are chosen randomly from the databases. Performance comparison of each time is repeated 100 times with different query points and their average is presented.

5.2 Discussion on Choosing SOM MapSize

As mentioned above, the number of map-points in the map layer, $MapSize$, must be decided in advance when SOM is used. If $MapSize$ is too large, there may be too many very-small ones among the discovered clusters. On the other hand, if $MapSize$ is too small, many very-big clusters may occur. Both these too cases can lengthen the time cost of data preprocessing and they also weaken the clustering degree of the objects in leaf nodes and then degrade search performance.

We performed many experiments to investigate influence of $MapSize$ on search performance and we observed that the best search performance is reached when

$$MapSize \approx \lceil \frac{2n}{M+m} \rceil \quad (1)$$

where n refers to the total number of objects in database; M,m, as mentioned above, refer to the maximum bound and the minium bound on the number of entries in each leaf node, respectively.

In Equation (1), $F_{ave} = (M+m)/2$ is the average number of the entries in a leaf node. According to the work[1], $m = 40\% * M$. If so, F_{ave} will be $70\% * M$. It is clear that $\lceil \frac{2n}{M+m} \rceil$ is the expected number of leaf nodes.

We call the $MapSize$ in Equation (1) S_{opt}. The experimental results with $MapSize \approx 0.5 * S_{opt}$, S_{opt} and $1.5 * S_{opt}$ are presented for comparison in this paper. In our experiments, M=10, m=4, S_{opt}=5714. Therefore, the experiment results with $MapSize = 60 * 60$ (=3600), $75 * 75$ (=5625) and $90 * 90$ (=8100) are presented in this paper.

5.3 Experimental Results

We cost about 1403 seconds for preprocessing (see Section 4.1) the database. The execution time and the number of object distance calculations (which is regarded as a very important factor on search performance) of NN search are tested and reported in this paper.

Figure 8 and **Figure 9** is the experiment result, where k refers to the number of neighboring objects to be retrieved.

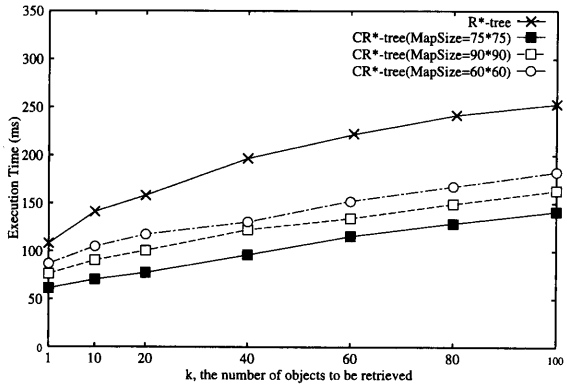

Fig.8 NN search performance (execution time) comparison.

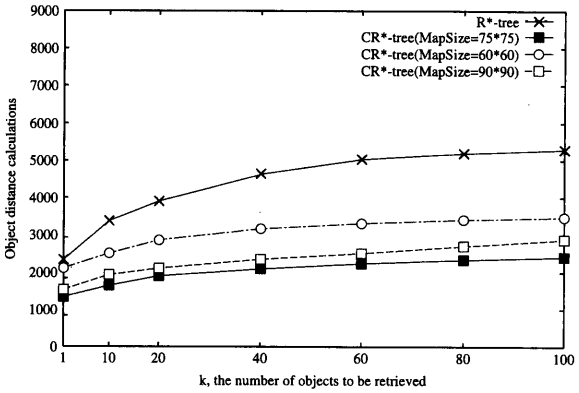

Fig.9 NN search performance (number of object distance calculations) comparison.

From the experimental results, we can observe that, if $MapSize$ is chosen properly, our proposal

has the lowest execution time, the smallest number of object distance calculations as compared to both R*-tree. We think this is because our proposal has the best clustering degree of the objects in leaf nodes, which affects greatly NN search performance.

6. Conclusion

In this paper, based on our observations on R*-tree we proposed an improved index structure for relatively static database by combining R*-tree with clustering technology to improve the clustering degree of objects in leaf nodes. Its NN search performance is greatly improved according to our analysis and experiments. Because the clustering technology is for static databases, our proposal is suitable for the applications with static or relatively static databases. Although our description, analysis and experiments in this study is based on R*-tree, we believe that the main idea of our proposal can also be used to many other members of R-tree family, including SS-tree[15], X-tree[2], SR-tree[3] and so on.

7. Future Work

In this paper, we used 12-dimensional image data to test NN search performance of our proposal. In the future, we will do the following works.

1. using some low-dimensional databases (e.g., GIS data) to test the behaviors of our proposal;
2. test the other search (e.g., range search) performance of our proposal;
3. compare search performance of our proposal with that of some other structures (e.g., packed R-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) S.Berchtold, D.Keim, H.P.Kriegel. "The X-tree: An Index Structure for High-Dimensional Data". In *Proceedings of VLDB*, 1996.
3) N. Katayama, S. Satoh. "The SR-tree: An index Structure for High-Dimensional Nearest Neighbor Queries". In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 369–380, May 1997.
4) G.R. Hjaltason, H. Samet. "Distance Browsing in Spatial Database". *ACM Transactions on Database Systems*, 24(2):265–318, June 1999.
5) K.S. Berchtold, C. Bohm, D. A. Keim, HP. Kriegel. "A Cost Model For Nearest Neighbor Search in High-Dimensional Data Space". In *Proceedings of PODS*, pages 78–86, Tucson, Arizona, 1997.
6) S. Guha, R. Rastogi and K. Shim. "CURE: An Efficient Clustering Algorithm for Large Databases". In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1998.
7) T. Zhang, R. Ramakrishnan and M. Livny. "BIRCH: An Efficient Data Clustering Method for Very Large Databases". In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1996.
8) J. Zavrel. *"Neural Information Retrieval"*. PhD thesis, University of Amsterdam, 1995.
9) A. Rauber. "LabelSOM: On the Labeling of Self-Organizing Maps". In *Proceedings of IJCNN'99*, Washington DC, July 1999.
10) T. Kohonen. "Self-Organization of Very Large Document Collections: State of the Art". In *Proceedings of ICNN98*, volume 1, pages 65–74, London, UK, 1998.
11) K. Oh, Y. Feng, K. Kaneko, A. Makinouchi. "SOM-Based R*-Tree for Similarity Retrieval". In *Proceedings of the Seventh International Conference on Database Systems for Advanced Applications (DASFAA)*, pages 182–190, Hongkong, May 2001.
12) Y. FENG, M. KUBO et al. A New SOM-based R*-tree: Building and Retrieving. *Research Reports on Information Science and Electrical Engineering of Kyushu University*, 6(2):209–214, 2001.
13) *H^2soft, http://www.h2soft.co.jp.*
14) C.E. Jacobs, A. Finkelstein, D.H. Salesin. "Fast Multiresolution Image Querying". In *Proceedings of SIGGRAPH95*, pages 6–11, Los Angeles, California, 1995.
15) D.A. White and R. Jain. "Similarity Indexing with the SS-tree". In *Proceedings of the 12th International Conference on Data Engineering*, pages 516–523, New Orleans, USA, Feb. 1996.