Enhancing R^*-tree to Improve Search Performance in OLAP Application

Feng, Yaokai
Department of Intelligent Systems, Faculty of Information Science and Electrical Engineering, Kyushu University

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

https://doi.org/10.15017/1515918

出版情報：九州大学大学院システム情報科学紀要. 9 (1), pp.1-6, 2004-03-26. 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9巻 第1号 平成16年3月

Research Reports on Information Science and
Electrical Engineering of Kyushu University
Vol.9, No.1, March 2004

Enhancing R*-tree to Improve Search Performance in OLAP Application

Yaokai FENG* and Akifumi MAKINOUCHI*

(Received December 11, 2003)

Abstract: In this study, first, it is pointed out that R*-tree can be applied to OLAP (On-Line Analytical Processing) application. And then, some features of OLAP databases are introduced. After we present that how the R*-tree is applied to OLAP field, we also introduce how to enhance the R*-tree to further improve the search performance in accordance with the features of OLAP application. This study is focused on ROLAP (Relational OLAP), one popular kind of the OLAP systems. Our proposals are discussed in details and examined by experiments using synthetic data. The experimental result indicates that our proposals can clearly improve the search performance.

Keywords: R*-tree, OLAP, SQL, Multidimensional search, OLTP

1. Introduction

There is increasing requirement for processing multidimensional range queries on business data usually stored in relational tables. For example, Relational On-Line Analytical Processing (ROLAP) in data warehouse is required to answer complex and various types of range queries on large amount of such data. In order to get good performance for such multidimensional range queries, multidimensional indices are helpful[1),2)].

Many index structures have been proposed in the last two decades. Among them, R*-tree[3)] is one of the well-known and successful ones and widely used in many applications and researches[4),5),6),7),8)]. In this study, the R*-tree is enhanced for indexing business data to improve the performance of multidimensional range queries on the business data. Note that our proposal can also be used to other members of the famous R-tree family.

In the works[4),9),8),10),11)], the aggregate values are pre-computed and stored in a multidimensional index as materialized view. The OLAP queries find aggregate values of data within a given range. When required, the aggregate values can be retrieved efficiently. In this study, we also use a multidimensional index for OLAP data. However, it is completely different from the related works in that our study focuses on using an enhanced R*-tree to speed up evaluation of range queries themselves.

In this study, first, it is pointed out that R*-tree can be applied to OLAP (On-Line Analytical Processing) application. And then, some features of OLAP databases are introduced. After we present that how the R*-tree is applied to OLAP field, we also introduce how to enhance the R*-tree to further improve the search performance in accordance with the feathers of OLAP application. This study is focused on ROLAP (Relational OLAP), one popular kind of the OLAP systems. Our proposals are discussed in detail and examined by experiments using synthetic data. Examination with TPC-H benchmark data is one of our future works.

2. Indexing Business Data Using R*-tree

Now, we briefly recall how the R*-tree index business data stored in a relational table and give some terms. Let T be a relational table with n attributes, denoted by $\mathrm{T}(A_1, A_2, \cdots, A_n)$. Attribute A_i $(1 \leq i \leq n)$ has domain $\mathrm{D}(A_i)$, a set of possible values for A_i. The attributes often have types such as boolean, integer, floating point, character string, date and so on. Each tuple t in T is denoted by $< a_1, a_2, \cdots, a_n >$, where a_i $(1 \leq i \leq n))$ is an element of $\mathrm{D}(A_i)$.

When the R*-tree is used in relational tables, some of the attributes are usually chosen as *index attributes*, which are used to build the R*-tree. For simplification of description, it is supposed without loss of generality that the first k $(1 \leq k \leq n)$ attributes of T, $< A_1, A_2, \cdots, A_k >$, are chosen as index attributes. Since the R*-tree can only deal with numeric data, an order-preserving transformation is necessary for each non-numeric index attributes.

After necessary transformations, the k index attributes form an k-dimensional space, called *index space*, where each tuple of T corresponds to one point.

* Department of Intelligent Systems

It is not difficult to find such a mapping function for boolean attributes and date attributes. For boolean data, "True" and "False" can be mapped onto 1 and 0, respectively, if "True"> "False" is assumed forcedly. This ordering has no practical problems, because the predicate of "equality" such as "A = True" or "A = False" is the only predicate pattern for the boolean attribute. Although implementation of "date" depends on DBMS, typical example of "date" in TPC-H benchmark consists of three integers representing year, month, and day. A simple function to get a numeric value for a "date" is to use the number of days from some reference date to this "date". In this paper, the day of Jan. 1, 1900 is used as the reference day, that is, the number of days from Jan. 1, 1900 to Apr. 5, 1998 is used to represent the date of Apr. 5, 1998.

It is not easy to map an arbitrary character string to a unique numeric data.The work[12] proposes an efficient approach that maps character strings to real numeric values within [0,1], where the mapping preserves the lexicographic order. This approach is also used in this study to deal with attributes of character string.

We call the value range of A_i, $[l_i, u_i]$ $(1 \leq i \leq k)$, *data range* of A_i attribute (in this paper, "dimension" and "index attribute" are used interchangeably). The length of the data range of A_i, $|u_i - l_i|$, is denoted by $R(A_i)$. The k-dimensional hyper-rectangle, $[l_1, u_1] \times [l_2, u_2] \times \cdots \times [l_k, u_k]$, forms the index space.

Simple but basic range queries are considered in the paper. The query condition is formed by chaining atomic predicates by logical "And". An atomic predicate represents an interval of a dimension like "$l \leq A \leq u$", where A is an attribute, l and u are range constants. The special case of "$l \leq A \leq l$" means "$A = l$". A range query on table $T(A_1, A_2, \cdots, A_n)$ is expressed by an SQL-like query language as follows.

```
Select ···
From T
Where $l_{q1} \leq A_{q1} \leq u_{q1}$
...
And $l_{qj} \leq A_{qj} \leq u_{qj}$
...
And $l_{qm} \leq A_{qm} \leq u_{qm}$
```

where $\{A_{q1}, \cdots, A_{qm}\} \subseteq \{A_1, \cdots, A_k\}$. Attributes specified in the range query condition is called *query attributes.*

3. R*-tree Used For OLAP Application

Because of the particularity of business data, some new features occur when the R*-tree is used to index business data.

As a feature of business data, the data ranges of the attributes are very different from each other. For instance, the data range of "Year" from 1990 to 2003 is only 13 while the amount of "Sales" for different "Product" may be up to several hundreds of thousands.

Another typical example of such domains with small cardinalities is boolean attribute, which has inherently only two possible values. Attribute with other data type may also semantically have small cardinality (e.g., day of the "week" with seven values). In LINEITEM table of TPC-H benchmark, RETURNFLAG, SHIPINSTRUCT, and SHIPMODE have only 3, 4, and 7 distinct values, respectively, although their data type is character string. These attributes cause inappropriate clustering pattern of the tuples among the R*-tree leaf nodes, which may deteriorates the search performance.

Now, two observations are presented as follows.

1) Imbalanced clustering.

Let us see the following example.

Table 1 YearlySales.

	Year	Product	Sales
t1	1999	"TV"	10,000,000
t2	2000	"VIDEO"	3,000,000
t3	2001	"CAMERA"	1,000,000

The length of data range in "YearlySales" dimension is very large (e.g., 9,000,000) while that in "Year" dimension is very small (e.g., only 14 from 1990 to 2003). According to our investigations, the MBR of each leaf node almost cover entire data range of Year dimension. This incurs fatal deterioration of range query performance. If only Sales dimension is specified as the query attribute, the query can restrict the nodes to be accessed, so it is evaluated more efficiently. On the other hand, if only Year attribute is specified in the range query condition like "Year = 1993", almost all nodes of the index have to be accessed to evaluate the queries. Thus, range query performance in this case depends on what attributes are used as query attributes.

Fortunately, the clustering pattern of the tuples among the R*-tree leaf nodes can be controlled, which will be discussed in detail later.

2) Many slender nodes exist.

Slender nodes means those having a very narrow side (even side length is zero) in some dimension. Some examples are those MBRs roughly shaped as a line segments in 2-dimensional spaces and roughly shaped as plane segments in 3-dimensional spaces.

The existing of slender nodes may leads to problems both with the R*-tree construction and with queries.

Let us consider the insertion algorithm of the R*-tree, using the example depicted in **Fig.1**. Point p is to be newly inserted. Certainly it should be inserted in Node B since it is nearer to Node B than to Node A. However, according to the insert algorithm of the R*-tree, p will be inserted to Node A in this case. This is because the area increment of doing so is smaller than that of inserting p to B. This will lead to a bad clustering of tuples among the leaf nodes, which greatly cut down the performance of queries.

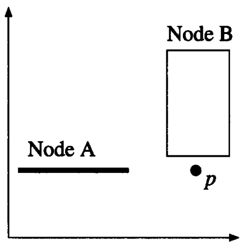

Fig.1 Slender nodes exist.

In all the range search algorithms, it is necessary to decide whether one node MBR and the query range intersect or not. The existing method to do so is to calculate the overlap volume between them. If one of them has the volume of zero, their overlap volume is zero and they are considered not intersected with each other even if the fact is contrary, which may lead to a wrong query result.

In addition, the range query performance with imbalanced clustering depends on what attributes are used as query attributes (discussed in Section 3.). That is, if some attributes are used in query, the query performance may be much worse than that of some others being used.

4. Enhancing R*-tree in Accordance with the Features of Business Data

In this section, we explain how to control the clustering pattern to improve range search performance and how to solve the problems of slender nodes.

4.1 Solving the Slender Nodes Problem

Extended normalization can improve the group performance of range queries. However it can not solve the problems of slender nodes. The reason is as follows. After normalization or extended normalization, the density of objects (or say tuples) along every dimension may become very different from each other. Thus, when the objects are inserted one by one to build the R*-tree, some dimension may be chosen as split axis very often. As a result, many slender nodes arise.

Our solution to the Problem of Slender Nodes is as follows.

The insert algorithm is revised. It is known that the insert algorithm of the R*-tree is a decisive factor to the clustering pattern of the objects among the leaf nodes, which greatly affect the query performance. The R*-tree use *area-criterion*, including area-enlargement and overlap-enlargement, to decide the subtree that the insert algorithm should follow next. However, this method has caused some problems, as discussed before, when the R*-tree is used on business data. In this study, a novel *distance-criterion* is introduced to settle this problem. When a new object is inserted, the distance-criterion is used first to decide which subtree should be followed next. Concretely speaking, the insert algorithm will recursively choose the child node having the nearest distance from the new object to follow. In the cases that more than one nodes have the nearest distance from the new object, the existing area-criterion is used.

4.2 Controlling the Tuples Clustering

It is well known that *normalization* is a common way to deal with the big difference among the data range in different dimensions. In the existing normalization, the attribute data are scaled so as to fall within a small range of [-1.0, 1.0] or [0.0, 1.0] in each index dimension[6),13)].

However, the existing normalization is too stiff; that is, all the index attributes are dealt with in the same way. In this study, *extended normalization* is used to control the clustering pattern according to requirement (e.g., according to importance degrees of the index attributes).

A point $(a_1, a_2, \cdots, a_k)$ in the index space is virtually mapped to

$$\left(\frac{a_1 - l_1}{R(A_1)} \times c(A_1), \cdots, \frac{a_k - l_k}{R(A_k)} \times c(A_k)\right),$$

where $(l_1, l_2, \cdots, l_k)$ is the left-lower corner of the index space , $R(A_i)$ $(1 \leq i \leq k)$ is the length of data range of A_i, and $c(A_i)$ $(1 \leq i \leq k)$ is *control coefficient* of A_i. The new normalized distance $Ndist(p_1, p_2)$ between two points $p_1 = (a_1, \cdots, a_k)$ and $p_2 = (b_1, \cdots, b_k)$ is defined as

$$Ndist(p_1, p_2) = \sqrt{\sum_{i=1}^{k} \left(\frac{b_i - a_i}{R(A_i)} \times c(A_i)\right)^2}.$$

While the existing normalization relocates virtually the data range of each dimension to [0.0, 1.0] or [-1.0, 1.0], the extended normalization relocates the data range of A_i $(1 \leq i \leq k)$ dimension to $[0, c(A_i)]$. Obviously, the existing normalization is a special case of the extended normalization when $c(A_i) = 1$ for $1 \leq i \leq k$. Data clustering among the leaf nodes will change along with the control coefficients varying. Our basic idea is, by selecting appropriate control coefficients for each dimension, to control the tuples clustering pattern among the leaf nodes and then to improve the total performance of a group of queries.

If the index attributes with larger control coefficients are used as query attributes, the number of index nodes to be accessed to evaluate the range query becomes smaller. This consideration leads to the idea that giving larger control coefficients to more important attributes may improve the total performance of range queries.

A simple idea to determine importance degree of each attribute is based on the number of its occurrences in the range conditions of the given query group. The more frequent some attribute is used, the bigger its importance degree is. The control coefficients of the attributes used in the index construction are roughly proportional to their importance degrees. Generally speaking, importance degree of each attribute is not necessarily proportional to the number of its occurrence if some attribute(s) need to be more emphasized. Anyway, it is not necessary to create a new data set for the extended normalization, which can be realized when the data are inserted in the index.

5. Experiments

We performed various experiments to show how much the range query performance is improved using our proposals. The page size in our system is 4KB and all the index structures are built based on "one node one page". To evaluate the performance of range queries we use average number of node accesses, which is common criterion for evaluation of search performance[12]. In OLAP field, attributes are generally categorized into two types[14]: *index attribute* (dimensions in index space) and *measure attributes* (whose values are often aggregated). The measure attribute is rarely specified as query attribute. This implies that a multidimensional index is built with all the attributes possibly used in queries.

5.1 Examination without Considering Execution Frequencies of Queries

Here, it is investigated how the tuples clustering controlled by the extended normalization affects the group performance of range queries. The synthetic tuples consist of 8 attributes $A_1, A_2, \cdots, A_8$, each of which is a floating point value uniformly distributed in the range of [0, 10000]. We use four of the eight attributes as index attributes. Each of leaf nodes contains all eight attributes and each of the other nodes contains the values of A_1, A_2, A_3, A_4. The total number of tuples is 1,000,000.

Table 2 Query group 1.

	A_1	A_2	A_3	A_4
query-1	○			
query-2	○	○		
query-3	○	○	○	
query-4	○	○	○	○
important degree	4	3	2	1

Table 3 Control coefficient used in each tree.

	$c(A_1)$	$c(A_2)$	$c(A_3)$	$c(A_4)$
Tree A	1	1	1	1
Tree B	4	3	2	1
Tree C	8	6	2	1

Table 2 shows query attributes specified in each query of *Query Group 1*. For example, in *query-2*, attributes A_1 and A_2 are used as query attributes. The last row of the table shows the importance de-

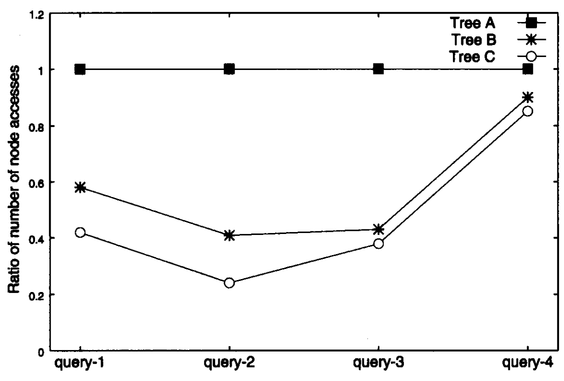

Fig.2 Relative performance of queries in group 1.

grees of index attributes, which is used to decide the control coefficients (discussed in Section 4.2). In this experiment the importance degree of each attribute is determined according to the number of its occurrences in the query group. For example, A_1 is specified in four queries in the group, while A_4 is specified only once. The indices used in this experiments are created using the new distance-criterion and the extended normalization. The three trees shown in **Table 3** of *Tree A, Tree B, and Tree C* are different in their control coefficients. Note that *Tree A* is normalized by "original normalization" and the control coefficients of *Tree B* are equal to the importance degrees of the query attributes. In *Tree C*, importance degrees of A_1 and A_2 are emphasized.

In the experiments, each query is executed 100 times with different intervals (selected randomly) for each predicate while the selectivity keeps fixed. The performance is measured in the average number of nodes accesses.

Figure 2 shows the relative performance to Tree A. In this experiment, the selectivity of each predicate is fixed to 1%. The numbers of node accesses of the Query Group 1 is showed in **Table 4**.

Table 4 Node Accesses of Query Group 1.

	Tree A	*Tree B*	*Tree C*
Total number of nodes	22953	22965	22990
query-1	2681	1544	1183
query-2	349	148	88
query-3	57	26	21
query-4	14	13	12
Total	3101	1741	1304

Figure 3 shows the result of another experiment

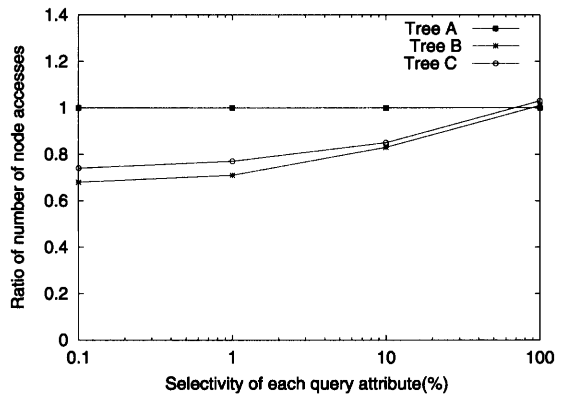

Fig.3 Group performance of queries in group 1 with varying selectivity.

using Query Group 1, where selectivity is changed from 0.1% to 100%. The performance is measured in terms of total number of node accesses of query group 1. Relative performance to Tree A is shown. This figure shows Tree B and Tree C outperform Tree A, especially when selectivity is small. When selectivity is too large, almost all nodes have to be accessed. Thus, the difference between Tree A and the others becomes small.

5.2 Examination with Considering Execution Frequencies of Queries

Table 5 Query group 2.

	A_1	A_2	A_3	A_4	*Frequency of execution*
query-1	○	○			7/15
query-2	○		○		4/15
query-3	○			○	1/15
query-4		○	○		1/15
query-5		○		○	1/15
query-6			○	○	1/15
importance degree	12	9	6	3	

Unlike Query Group 1, queries in Query Group 2 shown in **Table 5** have the same number of query attributes. The group is tested using the same set of tuples. Every query has two query attributes and the query group consists of 6 different queries. The frequency of each query being executed is considered to estimate the importance degree of each query attribute. The way to decide the importance degree for each index attribute is as follows.

See **Table 5**. The execution frequency of *query-1*, 7/15, means that *query-1* is executed 7 times if the total number of times of executing queries is

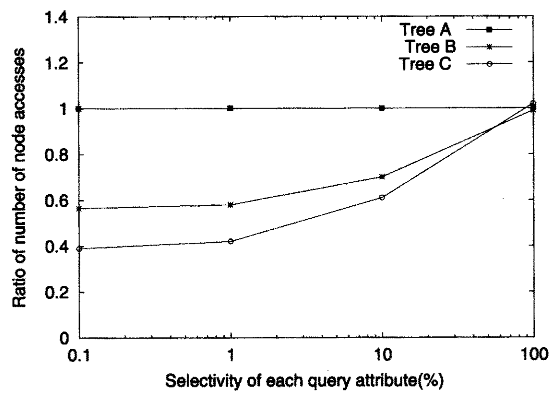

Fig.4 Group performance of queries in group 2 with varying selectivity.

15. This implies that the predicate concerning attribute A_1 like "$l_1' \leq A_1 \leq u_1'$" is evaluated 12 times in the 15 times of executions (shown in the last row) since attribute A_1 is specified in *query-1*, *query-2*, and *query-3*, whose execution frequencies are 7/15, 4/15, and 1/15, respectively. We let the importance degree of A_1 be 12, shown in the last row.

In this examination, total number of node accesses is measured while the selectivity of each query attribute changes from 0.1% to 100%. The importance degrees in Trees A and C are the same as the last examination, see **Table 3**. The importance degrees used in Tree B are determined considering the frequencies of executions as mentioned above.

Figure 4 shows the relative performance of Tree A, Tree B, and Tree C. This result also shows that Tree B and Tree C outperform Tree A.

6. Conclusions

In this study, first, it was pointed out that R*-tree can be applied to OLAP application. And then, some features of OLAP databases were introduced. After we presented that how the R*-tree is applied to OLAP field, we also introduced how to enhance the R*-tree to further improve the search performance in accordance with the feathers of OLAP application. This study was focused on ROLAP system, one popular kind of the OLAP systems. Our proposals are discussed in detail and examined by experiments using synthetic data.

7. Future works

The following two future works are being thought.

1. Examining our proposal with THC-H benchmark data.
2. Constructuring a new kind of multidimensional indices focusing OLAP databases.

References

1) V. Markl, F. Ramsak, and R. Bayer. Improving OLAP Performance by Multidimensional Hierarchical Clustering, 1999.
2) V. Markl, M. Zirkel, and R. Bayer. Processing Operations with Restrictions in Relational Database Management Systems without external Sorting, 1999.
3) N. Beckmann, and H. Kriegel. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles, 1990.
4) C. Chung, S. Chun, J. Lee, and S. Lee. Dynamic Update Cube for Range-Sum Queries, 2001.
5) D. Papadias, N. Mamoulis, and V. Delis. Algorithms for Querying by Spatial Structure, 1998.
6) H. Horinokuchi, S. Kuroki, and A. Makinouchi. Normalized R*-tree for Spatiotemporal Databases and Its Performance Tests, 1999.
7) H. P. Kriegel, T. Brinkhoff, and R. Schneider. Efficient Spatial Query Processing in Geographic Database Systems, 1993.
8) M. Jurgens, and H.-J. Lenz. The Ra*-tree: An Improved R-tree with Materialized Data for Supporting Range Queries on OLAP-Data, 1998.
9) Y. Kotidis, and N. Roussopoulos. An Alternative Storage Organization for ROLAP Aggregate Views Based on Cubetrees, 1998.
10) N. Roussopoulos, Y. Kotidis, and M. Roussopoulos. Cubetree: Organizaiton of and Bulk Incremental Updates on the Data Cube, 1997.
11) S. Hon, B. Song, and S. Lee. Efficient Execution of Range-Aggregate Queries in Data Warehouse Environments, 2001.
12) H. V. Jagadish, N. Koudas, and D. Srivastava. On Effective Multi-Dimensional Indexing for Strings, 2000.
13) J.Han, and M.Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, USA, 2001.
14) R. Agrawal, A. Gupta, and S. Sarawagi. Modeling Multidimesnional Databases, 1997.