Proposals to Improve Query Performance On Business Data

Yaokai Feng[♣], Akifumi Makinouchi[♣♣] and Kunihiko Kaneko[♣]

Abstract—**Multidimensional indices are efficient for improving the query performance on relational data. The R*-tree, a successful multi-dimensional index structure, has widely been used in research and DBMS products. The clustering pattern of the objects (i.e., tuples in relational tables) among R*-tree leaf nodes is one of the decisive factors on query performance. How then is the clustering pattern formed? In this paper, we first answer this question and then present several proposals to improve query performance on business data. Discussion and experimental result indicate that our new proposals are very efficient.**

Index Terms—**Multidimensional indices, R*-tree, clustering criterion, Multidimensional range query, TPC-H**

I. INTRODUCTION

More and more applications require the processing of multidimensional range queries on business data usually stored in relational tables. For example, Relational On-Line Analytical Processing in the data warehouse is required to answer complex and various types of range queries on large amounts of such data. In order to obtain good performance for such multidimensional range queries, multi-dimensional indices, in which the tuples are clustered among the leaf nodes to restrict the nodes to be accessed for queries, are helpful [15,16].

Many index structures exist. Among them, the R*-tree [9] is widely used in applications and research [1,3,4,5,8]. The R*-tree is also used in this study, although the proposals in the present study can also be applied to other hierarchical index structures.

In several previous studies [1,8,11,14,18], the aggregate values are pre-computed and stored in a multidimensional index as a materialized view. When required, the aggregate values can be retrieved efficiently. However, the present study is completely different from the related works in that we focus on enhancing the R*-tree in order to speed up the evaluation of range queries themselves.

In the present paper, the clustering pattern of tuples among the leaf nodes is reported as a decisive factor on search performance. In addition, there exist many slender leaf nodes when the R*-tree is used to index business data, which greatly degrades the query performance. Slender nodes refer to the nodes in which the Minimum Bounding Rectangles (MBRs) have at least one very narrow side (even zero) in some dimension(s). Some examples are MBRs that are roughly shaped as line segments in two-dimensional spaces and those that are roughly shaped as plane or line segments in three-dimensional spaces.

The present clustering criterion for constructing the R*-tree will be clarified to be unsuitable to business data. Several proposals are then presented in order to improve the query performance on business data. A discussion is presented and experimental results obtained in the present study indicate that the present proposals are very efficient.

The remainder of the paper is organized as follows. Section 2 describes the use of multidimensional indices for relational data. Section 3 presents our observations. Our proposals are presented in Section 4. Section 5 presents our experimental results, and Section 6 concludes the paper.

II. INDEXING BUSINESS DATA USING THE R*-TREE

In this section, we examine how to use the R*-tree to index business data and present several terms used in the present study.

Let T be a relational table with n attributes, denoted by T(A_1, A_2, ⋯, A_n). Attribute A_i ($1 \le i \le n$) has domain D(A_i), a set of possible values for A_i. The attributes often have types such as Boolean, integer, floating, character string, date, and so on. Each tuple t in T is denoted by <a_1,a_2, ⋯,a_n>, where a_i ($1 \le i \le n$) is an element of 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 ($1 \le k \le n$) attributes of T, <A_1,A_2, ⋯ ,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 attribute. After the necessary transformations, the k index attributes form a k-dimensional space, called the index space, where each tuple of T corresponds to one point.

[♣]The Graduate School of Information Science and Electrical Engineering, Kyushu University, Japan
{fengyk,keneko} @is.kyushu-u.ac.jp
[♣♣]Department of Information Network Engineering, Kurume Institute of Technology, Japan
akfumi@cc.kurume-it.ac.jp

It is not difficult to find such a mapping function for Boolean attributes and date attributes [17]. In a previous study [6], an efficient approach was proposed that maps character strings to real numeric values within [0,1], where the mapping preserves the lexicographic order. This approach is also used in the present study to deal with attributes of character string.

We call the value range of A_i, $[l_i, u_i]$ $(1 \le i \le k)$ *data range* of A_i, an index attribute (in the present 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. Attributes specified in the range query condition are called *query attributes*.

Simple but basic range queries are considered in the present paper. The query condition is formed by chaining atomic predicates by logical "AND". An atomic predicate represents an interval of a dimension such as "$l \le A \le u$", where A is an attribute, and l and u are range constants. The special case of "$l \le A \le l$" means "$A = l$". A range query on table T(A1, A2, …, An) is expressed by an SQL-like query language as follows.

Select …
From T
where $l_{q1} \le A_{q1} \le u_{q1}$
…
and $l_{qj} \le A_{qj} \le u_{qj}$
…
and $l_{qm} \le A_{qm} \le u_{qm}$

Here $\{A_{q1},\ldots,A_{qm}\} \subseteq \{A_1,\ldots,A_k\}$. Attributes specified in the range query condition are called *query attributes*.

If the R*-tree is used to index business data stored in a relational table, the all of the tuples are clustered in R*-tree leaf nodes. See Figure 1.

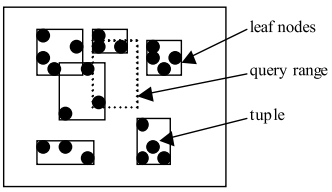

Figure 1. Leaf nodes and query range.

Figure 1 shows an example of leaf nodes and query range. The *query range*, given by the user, refers to the region in which the user wants to find the result.

Figure 1 shows that if the tuples are properly clustered among the leaf nodes, then the number of leaf nodes to be accessed for this range query will drop. Thus, the clustering pattern is one of decisive factors on query performance. However, the question as to who decides the clustering pattern remains. The answer is the "clustering criterion" in the insert algorithm of the R*-tree. Let us next examine the reason for this.

The R*-tree is constructed by inserting the objects one by one. In the constructing procedure, the insert algorithm has to choose a proper subtree to contain each new incoming tuple. In the present paper, the criterion that decides which subtree should be chosen is called the insert criterion or the clustering criterion. For a given dataset, this criterion decides the final clustering pattern of the tuples among the leaf nodes. Note that the present clustering criterion of the R*-tree cannot lead to a proper clustering pattern when the R*-tree is applied to business data. In addition, a novel clustering criterion and an extended normalization will be proposed.

III. OBSERVATIONS

This section describes our observations on the R*-tree used for business data.

A. Many Slender Nodes Exist

1) Business data distribution

As reported in our previous study [1], the data ranges of the attributes of business data differs greatly from each other. For instance, the data range of "Year" from 1990 to 2003 is only 14 whereas the amount of "Sales" for different types of "Product" may reach several hundreds of thousands. Another typical example of such attributes with small cardinalities is the Boolean attribute, which inherently has only two possible values. Attributes with other data types may also semantically have small cardinality (e.g., "Weekday" with seven values). In the LINEITEM table of the TPC-H benchmark, RETURNFLAG, SHIPINSTRUCT, and SHIPMODE, respectively, have only 3, 4, and 7 distinct values, although their data type is character string.

Figure 2 shows an example in two-dimensional space.

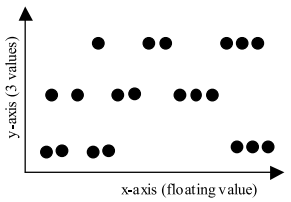

Figure 2. Tuples in index space.

In Figure 2, the y-axis has only three different values. In contrast, the x-axis shows many possible floating values. Thus, the tuples (black dots) are distributed in lines. It is not difficult to understand that the R*-tree built on business data with a such special distribution has many slender leaf nodes. Again, Slender nodes refer to nodes in which the MBRs have at least one very narrow side (even zero) in some dimension(s).

In order to investigate the existence of slender nodes in R*-tree used in business data, using the LINEITEM table in TPC-H benchmark [2], an R*-tree was constructed and all of the areas (or volumes) of the leaf nodes are computed. In total, 200,000 tuples are generated in this table, with 16 attributes.

Six attributes, SHIPDATE (date), QUANTITY (floating), DISCOUNT (floating), SHIPMODE (character string), SHIP-INSTRUCT (character string), and RETURNFLAG (character string), are selected as index attributes because they are often used as query attributes in the queries of the benchmark. The page size of our system is 4KB and each leaf node can contain at most 77 tuples. The R*-tree has four levels with 4,649 leaf nodes. We observe that, 2,930 of these 4,649 leaf nodes have 0-area, which is over 60%! In addition, there are many leaf nodes still have only very small areas. An example of the distribution of the leaf nodes is shown in Figure 3.

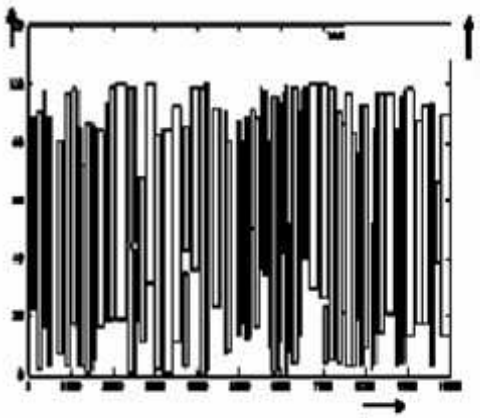

Figure 3. Distribution of the leaf nodes of the R*-tree used in business data.

2) Present clustering criterion

Since the clustering criterion is so important with respect to the clustering pattern of tuples among the leaf nodes of the R*-tree and the present study attempts to introduce a new clustering criterion, let us briefly recall the present clustering criterion of the R*-tree as follows.

A new tuple will be inserted in the node (subtree) at the current level with

1) (for leaf level only) the least enlargement of the overlapping area, if a tie occurs then

2) the least enlargement of the MBR area, if a tie occurs again then

3) the least MBR area.

This criterion means that, if the new tuple reaches the leaf level, for the newly incoming tuple, an attempt is made to enter every node, and the enlargement of the overlapping area in each case among the leaf nodes is calculated. And, the node with the least enlargement of the overlapping area is chosen to contain the newly incoming tuple. Then, if several nodes have the least enlargement, the enlargement of the MBR area in each case is calculated, and the node with the least enlargement of the MBR area is chosen. If another tie occurs, then the node with the smallest MBR area is chosen. If a tie occurs again, then one of the nodes with the smallest MBR area is chosen arbitrarily. For the non-leaf level, the enlargement of the overlapping area among the nodes is not calculated and only the above-mentioned 2) and 3) mentioned above in the criterion are used.

3) Existence of slender nodes is positive feedback

In this subsection, we will confirm that the existence of slender leaf nodes is a "positive feedback". That is, once some slender leaf nodes exist, the number of slender nodes will increase as the new tuples are inserted, which greatly deteriorates the search performance.

Let us consider the insertion algorithm of the R*-tree, using the example depicted in Figure 4(a). Node *A* is a slender node and point *p* is to be newly inserted. Point *p* should be inserted into Node *B* because it is much nearer than to Node *A*. However, according to the insert algorithm of the R*-tree, in this case *p* will be inserted into Node *A* because the area increment of doing so is smaller than that for inserting *p* into Node *B*. Even if the enlargement of the overlapping area among the nodes at this level is considered, Node *A* tends to be chosen. After *p* is inserted into Node *A*, Node *A* becomes very long, which may deteriorate the problem of the slender node.

Figure 4(b) shows another example. There are two MBRs that are shaped as line segments *A* and *B*. *p* is a new tuple to be inserted. Where should *p* be inserted? Intuitively, *p* should be inserted into Node *B*. However, *p* may also be inserted into Node *A*, although this enlarges the overlap (between Nodes *A* and *B*) and leads to a long node *A*. This is because the insertion algorithm of the R*-tree cannot determine which node, *A* or *B*, should be selected because both the overlapping area increment and the area increment for selecting *A* and selecting *B* are 0. As a result, either Node *A* or Node *B* may be selected as the default without consideration of the actual overlap. In addition, when a new point (tuple) with the same y-axis coordinate as *p* is inserted, the same process is repeated, and the new point is also inserted into the default node.

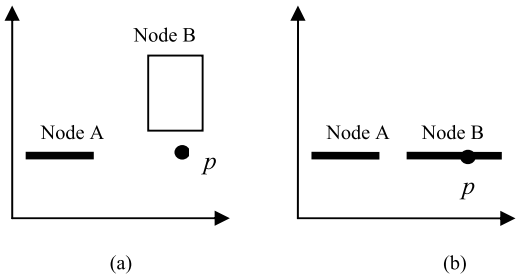

Figure 4. Insertion in the case that slender nodes exist.

In this way, newly incoming tuples tend to be inserted into existing slender nodes. The repeated insertion of such tuples leads to the overflow of slender nodes, and the slender nodes are split repeatedly. As a result,

1) many slender nodes are generated,

2) the space utilization of such nodes is greatly degraded, and the total number of nodes in the R*-tree tends to increase,

3) slender nodes tend to become long (there is a low band on the number of tuples in each leaf node),

4) overlapping among the leaf nodes is significant.

B. Problems Caused by Slender Nodes

The existence of slender nodes leads to some problems with both R*-tree construction and queries.

As discussed in Section 3.1.3, the existence of slender nodes is positive feedback, which means that, once slender nodes occur, such nodes are repeatedly produced. This greatly degrades the clustering pattern of tuples in the leaf nodes of the R*-tree, which greatly influences the search performance.

In all of the range search algorithms, it is necessary to decide whether one node MBR and the query range intersect. The existing methods involve calculating the overlap volume between the node MBRs and the query range. If one of the node MBRs has the volume of zero, then the overlap volume between the node MBR and the query range is zero and the node and the query range are considered not to intersect, even if the reality is otherwise, which may lead to an incorrect query result. In addition, the range query performance of the R*-tree with imbalanced clustering depends to a large extend on the attributes used in the actual queries.

Let us consider an example. The length of data range in the "Sales" dimension is very large (e.g., 5,000,000) while that in the "Year" dimension is very small (e.g., only 14 from 1990 to 2003). According to our investigations, the MBR of each leaf node cover almost the entire data range of the "Year" dimension. This incurs fatal deterioration of the range query performance. If only the "Sales" dimension is specified as the query attribute, then the query can restrict the nodes to be accessed, and so the query is evaluated more efficiently. On the other hand, if only the "Year" attribute is specified in the range query condition, for example, "Year = 1993", almost all of the nodes of the index have to be accessed in order to evaluate the queries.

Fortunately, we found that the clustering pattern of the tuples among the R*-tree leaf nodes can be controlled, which will be discussed in the next section.

IV. Proposals for the Application of the R*-tree to Business Data

Our proposals include a hybrid clustering criterion using a modified area calculation and an extended normalization scheme.

A. A Hybrid Clustering Criterion

Generally speaking, the present clustering criterion (as mentioned above) of the R*-tree is based on area, including the overlap area enlargement, the MBR area enlargement, and the MBR area, which leads to many slender nodes and significant overlap among the leaf nodes. In this section, we explain how to deal with the problem of slender nodes by using a hybrid clustering criterion.

1) Modifying the area calculation.

Why can a proper subtree or leaf node not be found for newly incoming tuples? The answer is that the enlargements both in the overlap area and in the MBR area are zero for 0-area nodes. Thus, comparison can not be made reasonably.

In order to avoid this situation, we modified the area calculation. When the area of a rectangle (a node MBR or the overlap region of two node MBRs) is calculated, then all of the zero-sides (i.e., the side length is zero), if exist, of this rectangle are set to a trivial non-zero positive value (e.g., 10^{-4} in our experiments).

The original area calculation of rectangle R is:

$$Area(R) = \prod_{i=1}^{d} S_i,$$

where S_i is the side length of R in dimension i. and d is dimensionality of the index space.

In the present study, this equation is modified as:

$$Area'(R) = \prod_{i=1}^{d} S_i',$$

$$S_i' = \begin{cases} trivial-value & S_i = 0, \\ S_i & otherwise, \end{cases}$$

where the trivial-value is set to 10^{-4}. This trivial value must be less than the unit in this attribute in order to avoid confusing. At the same time, the trivial-value should not be too small, in order to prevent the situation in which the calculation result cannot be expressed. These two conditions are not difficult to guarantee in actual applications. In this way, most non-comparable situations caused by 0-area nodes can be avoided. Note that, this modification only changes the clustering pattern of tuples among the leaf nodes and has no effect on the correctness of the query result.

2) Introducing a distance-criterion.

As mentioned below, a distance-based clustering criterion is introduced to the existing area-based criterion.

1) For the leaf level, compare the enlargements of overlap areas using the modified calculation. If a tie occurs, then

2) Compare the enlargements of MBR areas using the modified calculation. If a tie occurs, then

3) Choose the nearest subtree (a leaf node for the leaf level).

Next, let us examine how to calculate the distance from one point to a rectangle region in a d-dimensional space.

For a point $p = (p_1, \cdots, p_d)$ and a rectangle R. Let the points $s = (s_1, \cdots, s_d)$ and $t = (t_1, \cdots, t_d)$ be the two vertices of the node MBR with the minimum coordinates and maximum coordinates in each axis, respectively. The distance from p to R, dist(p, R), can be given by

$$dist(p, R) = \sqrt{\sum_{i=1}^{d} |p_i - r_i|^2},$$

where

$$r_i = \begin{cases} s_i & p_i < s_i, \\ t_i & p_i > t_i, \\ p_i & otherwise. \end{cases}$$

B. Extended Normalization

Normalization is a common way to deal with 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 [4,7]. However, the existing normalization is too stiff. That is, all of the index attributes are dealt with in the same way. In the present study, *extended normalization* is used to control the clustering pattern. In the extended normalization, the values of each attribute are normalized independently, that is, different attributes many be normalized to different ranges.

Let us assume that the index attributes are (A_1, A_2, …, A_k), the original data ranges of which are $R(A_1)$, $R(A_2)$, …, $R(A_k)$, respectively. Using the extended normalization, the data values of A_i ($1 \le i \le k$) are normalized to [0, c_i], where c_i reflects the degree of importance of A_i. The degrees of importance of attributes can be decided by users according to, for example, the frequency of the attributes occurring in the actual queries.

Thus, each tuple in the original index space is mapped to the normalized index space. In addtion, the point (tuple) (a_1,a_2,…,a_k) in the original index space is mapped to

$$\left(\frac{a_1 - l_1}{R(A_1)} \times c_1, \frac{a_2 - l_2}{R(A_2)} \times c_2, \cdots, \frac{a_k - l_k}{R(A_k)} \times c_k \right),$$

where l_i ($1 \le i \le k$) refers to the minimum value of A_i. Here, c_i ($1 \le i \le k$) is also called the control coefficient of A_i. Thus, the normalized distance $Ndist(p1,p2)$ between two points p_1(a_1,…,a_k) and p_2(b_1,…,b_k) can be calculated as

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

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 the A_i ($1 \le i \le k$) dimension to [0,c_i]. The existing normalization is a special case of the extended normalization when c_i= 1 for $1 \le i \le k$. The clustering pattern of the tuples among the leaf nodes will change along with the variation of the control coefficients. The basic idea is that by selecting appropriate control coefficients for each dimension, to control the clustering pattern of the tuples among the leaf nodes and then improve the query performance of the often-used queries. If the index attributes with larger control coefficients are used as query attributes, the number of index nodes to be accessed for this query becomes smaller. This consideration leads to the idea that providing larger control coefficients to more important attributes may improve the query performance of the often-used queries. A simple idea to determine the degree of importance of an attribute is based on its occurrence frequency in the actual queries. The more frequently an attribute is used, the greater its degree of importance. The control coefficients of the attributes used in the index construction are roughly proportional to their degrees of importance. Note that, (1) if some attribute needs to be emphasized further, then its degree of importance is not necessarily proportional to the frequence of its occurrence, and (2), importantly, it is not necessary to create a new data set for the extended normalization, which can be realized when the data are inserted into the index.

The main purpose of the present study is to decrease the overlap among leaf nodes and control the clustering pattern of the tuples in order to improve the query performance of the important queries. At the same time, the number of slender nodes is greatly decreased and the total space utilization of nodes can also be improved.

V. Experiments

Using the TPC-H benchmark data [2] and Zipf distributed data (the Zipf constant is 1.5 like the works [19,20]), we performed various experiments to show the degree to which the query performance can be improved using our proposals.

Dataset and index attributes of TPC-H Benchmark data: The Lineitem table of the TPC-H benchmark, which has 16 attributes of various data types including floating, integer, date, string, and Boolean, is used. The table used in our experiments has 200,000 tuples. Six of the total 16 attributes are chosen as index attributes, including SHIPDATE (date), QUANTITY (floating), DISCOUNT (floating), SHIPMODE (character string), SHIPINSTRUCT (character string), and RETURNFLAG (character string), because they are often used as query attributes in the queries of the benchmark.

Dataset with Zipf distribution: we created a dataset having 200,000 points with Zipf distribution in a six-dimensional space. The Zipf constant is 1.5 like the work [19, 20]. The Zipf distributed data are often used in OLAP researches.

Queries on the TPC-H benchmark data: The query ranges of QUANTITY (floating) and DISCOUNT (floating) are both changed from 10% to 100% of their entire data ranges. As for the date attribute of SHIPDATE (date), the query range is the period of one year and it is selected randomly for each query. One value is chosen randomly in each of the other three attributes (string), because the numbers of possibly different values are only 3, 4, and 7, respectively.

Queries on the Zipf data: the queries on the six-dimensional Zipf data are relatively simple. All of the dimensions are handled equally. For the sake of simplicity without loss of generality, all of the query ranges are cubes. That is, for each query, the query lengths are the same in all query dimensions. And the side-length changes from 10% to 100%.

Each query with the query range of the same size is repeated 100 times for different locations (randomly), and the average numbers of accessed nodes are presented. The average number of node accesses is a common criterion for evaluating query performance [6]. The page size in our system is 4KB and all the index structures are built based on “one node one page”.

A. Effect of the Hybrid Clustering Criterion

In order to investigate the effect of the new clustering criterion on the R*-tree itself, the total numbers of nodes in R*-trees with different clustering criteria are presented in Table 1, where M refers to the upper bound on the number of tuples contained in each leaf node.

Table 1. R*-tree with different clustering criteria.

	Original criterion	Hybrid criterion
M	77	77
Height	4	4
Nodes (TPC-H)	4892	3783
Nodes (Zipf)	4364	3513

From Table 1, we can observe that the hybrid clustering criterion makes the R*-tree more compact for both TPC-data and Zipf data..

Table 2-1 presents a comparison of the number of different accessed nodes for TPC-H data. The query range refers to the side length of the query range in the two floating attributes, i.e., QUANTITY and DISCOUNT. Table 2-2 is a similar comparison using Zipf data.

Tables 2-1 and 2-2 show that the hybrid clustering criterion can greatly improve the query performance both for TPC-H data and the Zipf data. We also considered the fact that, for the TPC-H data, the number of accessed nodes does not always increase as the query range grows because that the query ranges in the other four index attributes change randomly at the same time.

Table 2-1. Number of accessed nodes using TPC-H data.

Query range	Original R*-tree	R*-tree with *hybrid criterion*
10%	369	95
20%	648	126
30%	603	131
40%	388	137
50%	683	237
60%	489	248
70%	708	231
80%	691	275
90%	571	357
100%	764	358

Table 2-2. Number of accessed nodes using Zipf data.

Query range	Original R*-tree	R*-tree with *hybrid criterion*
10%	25	21
20%	162	134
30%	299	153
40%	413	192
50%	612	326
60%	926	413
70%	1320	922
80%	2468	1501
90%	3096	2195
100%	4364	3513

B. *Effect of Extended Normalization*

Using the above-mentioned TPC-H table and Zipf distributed data, the effect of the extended normalization is investigated. We can not exhaust all possible patterns of range queries using six index attributes and so select range queries using two or three index attributes. Two groups of range queries are tested as examples.

Table 3 shows the attributes and their corresponding dimensions for TPC-H data. Tables 4 and 5 are the two groups of queries.

Table 4 is Query Group A, consisting of five range queries, each of which has two or three query attributes. Table 5 shows Query group B, consisting of four range queries. Unlike Query Group A, each query in Query Group B has the same number of query attributes. The degree of importance given to each attribute is based on the number of its occurrences in the queries.

Table 3. Attributes and their dimensions (for TPC-H data).

Attributes	Dimensions
SHIPDATE	A1
QUANTITY	A2
SHIPMODE	A3
SHIPINSTRUCT	A4
DISCOUNT	A5
RETURNFLAG	A6

Table 4. Query group A

	A1	A2	A3	A4	A5	A6
Query-1	○	○			○	
Query-2	○		○			
Query-3		○		○		○
Query-4	○			○		
Query-5		○	○			
Importance Degree	3	3	2	2	1	1

Table 5: Query group B.

	A1	A2	A3	A4	A5	A6
Query-1		○	○	○		
Query-2	○		○		○	
Query-3	○	○				○
Query-4	○	○		○		
Importance Degree	3	3	2	2	1	1

The performance comparison on TPC-H data is shown in Tables 6-1, and 7-1. The same comparison using Zipf data is shown in Tables 6-2 and 7-2. “without hybrid criterion” in these tables indicates that the R*-tree is built using the extended normalization, but not the hybrid clustering criterion, and “with hybrid criterion” indicates that the R*-tree is built using both the extended normalization and the hybrid clustering criterion. The query range in each of the float dimensions is 10% of the entire data range and only one value is selected for each of the string dimensions.

These results show that, except *Query-1* in the group A and *Query-3* in the group B for TPC-H data, *Query-1,3* in the group

A and *Query-3* in the group B for Zipf data, all queries perform better using our proposals. In addition, the total performance of both Query Group A and Query Group B has been clearly improved.

Table 6-1. Performance comparison of group A (TPC-H data).

	Original R*-tree	R*-tree using extended normalization	
		Without hybrid criterion	With hybrid criterion
Query-1	392	802	668
Query-2	2701	2532	1997
Query-3	2866	1049	870
Query-4	2638	2415	2063
Query-5	4176	2566	2318
Total	12773	9364	7916

Table 6-2. Performance comparison of group A (Zipf data).

	Original R*-tree	R*-tree using extended normalization	
		Without hybrid criterion	With hybrid criterion
Query-1	58	103	84
Query-2	847	616	276
Query-3	625	802	767
Query-4	763	518	381
Query-5	1018	901	794
Total	3311	2940	2302

Table 7-1: Performance comparison of group B. (TPC-H data)

	Original R*-tree	R*-tree using extended normalization	
		Without hybrid criterion	With hybrid criterion
Query-1	3950	1209	959
Query-2	2266	869	861
Query-3	391	1004	590
Query-4	469	977	535
Total	7076	4059	2945

Table 7-2: Performance comparison of group B. (Zipf data)

	Original R*-tree	R*-tree using extended normalization	
		Without hybrid criterion	With hybrid criterion
Query-1	732	625	506
Query-2	673	721	634
Query-3	695	753	712
Query-4	778	615	508
Total	2878	2714	2360

VI. CONCLUSIONS

In the field of OLAP, it is important to process various types of range queries on business data. The R*-tree is one of the successful multidimensional index structures. In the present paper, we attempted to enhance the R*-tree in order to evaluate range queries on OLAP data more efficiently. Our proposals include a hybrid clustering criterion using a modified area calculation and an extended normalization scheme. The experiment results of the present study indicate that our proposals are very efficient.

ACKNOWLEDGEMENT

This research is partially supported by Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research 15650017 and 16200005. The authors also would like to thank Mr. Zhibin Wang, who conducted part of the experiments.

REFERENCES

[1] C. Chung, S. Chun, J. Lee, and S. Lee (2001). Dynamic Update Cube for Range-Sum Queries. Proc. VLDB Intl. Conf.,

[2] Council (1999). TPC benchmark H standard specification (decision support) ".*http://www.tpc.org/tpch/*

[3] D. Papadias, N. Mamoulis, and V. Delis (1998). Algorithms for Querying by Spatial Structure. Proc. VLDB Intl. Conf.

[4] H. Horinokuchi, and A. Makinouchi (1999). Normalized R*-tree for Spatiotemporal Databases and Its Performance Tests. IPSJ Journal, Vol. 40, No. 3.

[5] H. P. Kriegel, T. Brinkhoff, and R. Schneider (1993). Efficient Spatial Query Processing in Geographic Database Systems.

[6] H. V. Jagadish, N.Koudas, and D. Srivastava (2000). On Effective Multi-Dimensional Indexing for Strings. Proc. ACM SIGMOD Intl. Conf.

[7] J. Han and M. Kamber (2001). Data Mining—Concepts and Techniques. Morgan Kaufmann press.

[8] M. Jurgens, and H.-J. Lenz (1998). The Ra*-tree: An Improved R-tree with Materialized Data for Supporting Range Queries on OLAP-Data. Proc. DEXA Workshop.

[9] N. Beckmann, and H. Kriegel (1990). The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. Proc. ACM SIGMOD Intl. Conf.

[10] N. Roussopoulos, S.K and F. Vincent (1995). Nearest neighbor Query. Proc. ACM SIGMOD Intl. Conf.

[11] N. Roussopoulos, Y. K and M. Roussopoulos (1997). Cubetree: Organizaiton of and Bulk Incremental Updates on the Data Cube. Proc. ACM SIGMOD Intl. Conf.

[12] R. Agrawal, A. Gupta, and S. Sarawagi (1997). ModelingMultidimesnional Databases. Proc. Intl. Conf. on Data Engineering (ICDE).

[13] S. Hon, B. Song, and S. Lee (2001). Efficient Execution of Range-Aggregate Queries in Data Warehouse Environments. Proc. the 20th Intl. Conf. on conceptual modeling.

[14] S. Hong, B. Song and S. Lee (2001). Efficient Execution of Range-Aggregate Queries in Data Warehouse Environments, Proc. 20th international Conference on CONCEPTUAL MODELING (ER 2001).

[15] V. Markl, F. Ramsak, and R. Bayer (1999a). Improving OLAP Performance by Multidimensional Hierarchical Clustering. Proc. IDEAS Intl. Synposium.

[16] V. Markl, M. Zirkel, and R. Bayer (1999b). Processing Operations with Restrictions in Relational Database Management Systems without external Sorting. Proc. Intl. Conf. on Data Engineering.

[17] Y. Feng, A. Makinouchi, and H. Ryu (2004). Improving Query Performance on OLAP-Data Using Enhanced Multidimensional Indices. Proc. ICEIS Intl. Conf.

[18] Y. Kotidis, and N. Roussopoulos (1998). An Alternative Storage Organization for ROLAP Aggregate Views Based on Cubetrees. Proc. ACM SIGMOD Intl. Conf.

[19] S. Hong, B. Song, and S. Lee (2001). Efficient Execution of Range Aggregate Queries in Data Warehouse Environments. Proc. Intl. Conf. on the Entity Relationship Approach (ER).
[20] C. Zhang, J. Naughton, et. al.: On Supporting Containment Queries in Relational Database Management Systems. Proc. ACM SIGMOD Intl. Conf.

Yaokai Feng received the B.S. and M.S. degrees in Computer Science in 1986 and 1992, respectively. Since he received Ph.D degree in Information Science from Kyushu University, Japan, in 2003, he has been with in the same university as an assistant professor. He is a member of IPSJ, IEEE, ACM and an editorial board member of IAENG International Journal of Computer Science.

Akifumi Makinouchi received his B.E. degree from Kyoto University, Japan, in 1967, Docteur-ingereur degree from Univrcite de Grenoble, France, in 1979, and D.E. degree from Kyoto University, Japan, 1988. Since 1990, he has been with the Graduate School of Information Science and Electrical Engineering, Kyushu University, Japan, where he is a professor. He is a member of IPSJ, ACM, and IEEE and fellow of IEICE.

Kunihiko Kanako received his Ph.D. degree form Kyushu University. Since 1996, he has been with the Graduate School of Information Science and Electrical Engineering, Kyushu University, Japan, where he is an associate professor. His research interests include spatial databases, and biomedical databases. He is a member of IPSJ, IEICE, ACM, and IEEE