IMPROVING QUERY PERFORMANCE ON OLAP-DATA USING ENHANCED MULTIDIMENSIONAL INDICES

Yaokai Feng, Akifumi Makinouchi, and Hiroshi Ryu
Graduate School of Information Science and Electrical Engineering, Kyushu University, Fukuoka City, Japan

Keywords: OLAP, multidimensional index, R*-tree, range query

Abstract: Multidimensional indices are efficient to improve the query performance on OLAP data. As one multi-dimensional index structure, R*-tree is popular and successful, which is a member of the famous R-tree family. We enhance the R*-tree to improve the performance of range queries on OLAP data. First, the following observations are presented. (1) The clustering pattern of the tuples (of the OLAP data) among the R*-tree leaf nodes is a decisive factor on range search performance and it is controllable. (2) There often exist many slender nodes when the R*-tree is used to index OLAP data, which causes some problems both with the R*-tree construction and with queries. And then, we propose an approach to control the clustering pattern of tuples and propose an approach to solve the problems of slender nodes, where slender nodes refer to those having a very narrow side (even the side length is zero) in some dimension. Our proposals are examined by experiments using synthetic data and TPC-H benchmark data.

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. A typical ROLAP range query is as follows. "**Select** sum (EXTENDEDPRICE* DISCOUNT) **From** *LINEITEM* **Where** QUANTITY ≤ 25 and 0.1 ≤ DISCOUNT≤0.3 and 2001-01-01≤SHIPDATE ≤ 2001-12-31", where *LINEITEM* is a table having sixteen attributes used in TPC-H benchmark (Council, 1999). In this query, three attributes QUANTITY, DISCOUNT, and SHIPDATE form the range condition. In order to get good performance for such multidimensional range queries, multidimensional indices are helpful (V. Markl and Bayer, 1999a; V. Markl and Bayer, 1999b), in which the tuples are clustered among the leaf nodes to restrict the nodes to be accessed for a query.

Many index structures have been proposed in the last two decades. Among them, R*-tree (Beckmann and Kriegel, 1990) is one of the well-known and successful ones and widely used in many applications and researches (C. Chung and Lee, 2001; D. Papadias and Delis, 1998; H. Horinokuchi and Makinouchi, 1999; H. P. Kriegel and Schneider, 1993; Jurgens and Lenz, 1998). 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 (C. Chung and Lee, 2001; Kotidis and N. Roussopoulos, 1998; Jurgens and Lenz 1998; N. Roussopoulos and Y. Kotidis, 1997; S. Hon and Lee, 2001), 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 paper, first, it is pointed out that, when the R*-tree is used for indexing business data, the clustering pattern of tuples among the leaf nodes is a decisive factor on range search performance. Then, we explain the clustering pattern is controllable and show how to control it to improve the group performance of range queries, i.e., to

Feng Y., Makinouchi A. and Ryu H. (2004).
IMPROVING QUERY PERFORMANCE ON OLAP-DATA USING ENHANCED MULTIDIMENSIONAL INDICES.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 282-289
DOI: 10.5220/0002632602820289

improve the total performance of evaluating a group of range queries issued on a relational table. Meanwhile, we also point out that there often exist many very slender leaf nodes when the R*-tree is used to index business data, which causes some problems both with the R*-tree construction and with queries, where *Slender nodes* means those having a very narrow side (even the side length is zero) in some dimension. We also propose an approach to solve the problems of slender nodes.

The rest of the paper is organized as follows. Section 2 introduces the R*-tree and describe how to use multidimensional indices for relational table. Section 3 presents our new observations when the R*-tree is used to business data. Section 4 describes our proposal. Section 5 gives experimental comparison using synthetic data as well as realistic data, and Section 6 concludes the paper.

2 R*-TREE IN BUSINESS DATA

This section gives a brief review of the R*-tree and describe how to use it to business data.

2.1 R*-tree

Let us briefly recall the R*-tree.

The R*-tree (Beckmann and Kriegel, 1990) is a hierarchy of nested d-dimensional MBRs (Minimum Bounding Rectangles). Each non-leaf node of the R*-tree contains an array of entries, each of which consists of a pointer and an MBR. The pointer refers to one child node of this non-leaf node and the MBR is the minimum bounding rectangle of the child node referred to by the pointer. Each leaf node of the R-tree contains an array of entries, each of which consists of an object identifier and its corresponding point (for point-object databases) or its MBR (for extended-object databases). The R*-tree is built by inserting the objects (tuples for relational tables) one by one. Throughout the remainder of this paper, no distinction is made between R*-tree nodes and their corresponding MBRs in the corresponding multidimensional space when the meaning is clear in the context. Also, the terms of tuple and object are also used interchangeably.

The R*-tree is one of the most successful variants of the well-known R-tree family. It uses sophisticated insertion and node splitting algorithms with the forced reinsertion mechanism.

2.2 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 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 point, 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 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.

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 (H. V. Jagadish and Srivastava, 2000) 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] ($1 \le i \le k$), *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, $[l_1,u_1]\times[l_2,u_2]\times\ldots\times[l_k,u_k]$, forms the index space.

Let us see the following example.

Table 1: YearlySales

	Year	Product	Sales
t1	1999	"TV"	6,000,000
t2	2000	"VIDEO"	3,000,000
t3	2001	"CAMERA"	1,000,000

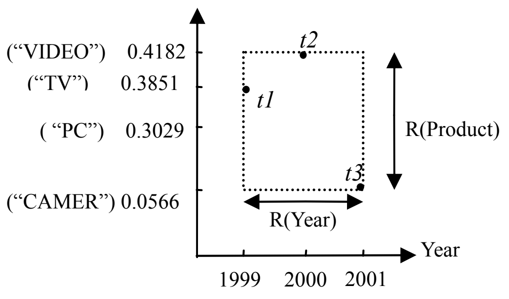

Figure 1: Example of index space

Figure 1 shows the index space of Table 1, where the data types of Year, Product, and Sales are integer, character string, and long integer, respectively. The first two attributes are used as index attributes, which form 2-dimensional index space of "Year-Product". The attribute values of Product are mapped to floating point data using the approach (H. V. Jagadish and Srivastava, 2000). The lengths of the data ranges are R(Year)=2 and R(Product)}=0.3616.

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 "$l \le A \le u$", where A is an attribute, l and u are range constants. The special case of "$l \le A \le l$" means "$A = l$". A range query on table $T(A_1,A_2,\ldots,A_n)$ is expressed by an SQL-like query language as follows.

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

Select …
From T
Where $l_{q1} \le A_{q1} \le u_{q1}$
…
And $l_{qj} \le A_{qj} \le u_{qj}$
…
And $l_{qm} \le A_{qm} \le u_{qm}$

3 OBSERVATIONS ON R*-TREE USED FOR OLAP DATA

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.

3.1 Imbalanced Clustering Deteriorates Range Search Performance

Let us see the above example again.

The length of data range in "Sales" dimension is very large (e.g., 5,000,000) 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.

3.2 The Problems of Slender Nodes

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 line segments in 2-dimensional spaces and roughly shaped as plane segments in 3-dimensional spaces.

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

3.2.1 Problem with R*-tree Construction

Let us consider the insertion algorithm of the R*-tree, using the example depicted in Figure 2 (a). 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 Node B. This will lead to a bad clustering of tuples among the leaf nodes, which greatly cut down the performance of queries.

Let us to see another case shown in Figure 2 (b). There are two MBRs shaped as line segments, A and B. Let assume *p* is a point to be inserted. Intuitively, *p* should be included in Node B whose MBR is a line segment. Actually, *p* may be inserted in Node A, although this may enlarge the overlap of A and B. This is because the insertion algorithm of the R*-tree cannot determine which node, A or B, should be selected since both volume increment and overlap increment of selecting A and selecting B are 0. As a result, either Node A or Node B is selected as default without consideration of actual overlap. Here, we assume that Node A is selected. When a new point with the same coordinate of A2 dimension as *p* is inserted again, the same process is repeated and the point is also inserted into Node A. The repeated insertion of such points leads to the overflow of Node A. The node is split into Node A and a new node, say Node A″. Repeated insertions

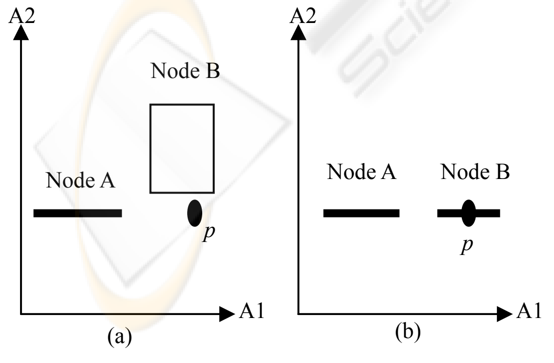

Figure 2: Slender nodes exist

of points like *p* leads to node A splitting again, which generate a new Node A″, and so on. As a result, the space utilization of such nodes degrades and the total number of nodes tends to increase. Moreover, the heavy overlaps among the leaf nodes also greatly influence the search performance.

3.2.2 Problem with Range Query

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 3.1). 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 FOR OLAP 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 Controlling the Tuples Clustering Pattern to Improve Range Search Performance

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 (H. Horinokuchi and A. Makinouchi, 1999; J. Han and Kamber, 2001).

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 $(a_1,a_2,\ldots,a_k)$ 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 $(l_1,l_2,\ldots,l_k)$ is the left-lower corner of the index space , $R(A_i)$ $(1\le i\le k)$ is the length of data range of A_i, and $c(A_i)$ $(1\le i\le k)$ is *control coefficient* of A_i. The new normalized distance $Ndist(p_1,p_2)$ between two points $p_1(a_1,\ldots,a_k)$ and p_2 $(b_1,\ldots,b_k)$ 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 $(1\le i\le k)$ dimension to $[0,c(A_i)]$. Obviously, the existing normalization is a special case of the extended normalization when $c(A_i)=1$ for $1\le i\le k$. 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 group performance of range queries, where the group performance refers to the total performance of a group of range 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 an attribute is based on the number of its occurrences in the range conditions of the given query group. The more frequent it 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. Importance degree of an 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.

4.2 Solving the Problems of Slender Nodes

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 Problems of Slender Nodes includes the following measures.

1. Revising the insert algorithm of the R*-tree. 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 of nodes, to decide the sub-tree 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 to the R*-tree, the distance-criterion is used first to decide which sub-tree 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 case that more than one node have the nearest distance from the new object, the existing area-criterion is used. Let the points $s=(s_1,\ldots,s_n)$ and $t=(t_1,\ldots,t_n)$ be the two vertices of the node MBR with the minimum coordinates and maximum coordinates in each axis, respectively. The distance of a node $v(s,t)$ from point $p=(p_1,\ldots,p_n)$ is given by:

$$dist(v,p)=\sqrt{\sum_{i=1}^{n}\left|p_i-r_i\right|^2}\,,$$

where

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

Figure 3 is an example. If the existing area-criterion is used in the insertion algorithm, the new object p is inserted in Node A. If the distance-criterion is used in this case, p is inserted in Node B, which, obviously, leads to a better clustering of the tuples among the leaf nodes. Again, the clustering of the tuples among the leaf nodes is a decisive factor on the search performance.

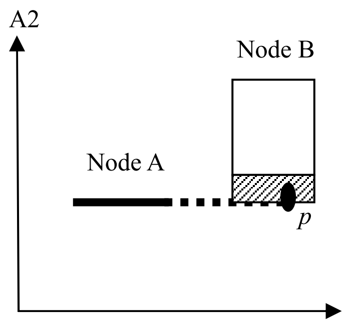

Figure 3: Choosing node to follow in the insert algorithm.

2. When deciding whether one node and the query range intersect or not, we check if they overlap in every dimension, instead of calculating the overlap area between them. Overlap area of one node MBR and the query range can not be used in this case since a node MBR and the query range may intersect even if their overlap area is zero.

5 EXPERIMENTS

We performed various experiments to show how much the range query performance is improved using our proposals. Due to the limitation of space, only the result of the examination using realistic data based on TPC-H benchmark is presented. 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 a common criterion for evaluation of search performance (H. V. Jagadish and Srivastava, 2000). In OLAP field, attributes are generally categorized into two types (R. Agrawal and Sarawagi, 1997): *index attributes* (dimensions in index space) and measure *attributes* (whose values are often aggregated).

We examine our proposals using the following four R*-tree based structures. (1) *Original R*-tree*. (2) *AR*-tree*: created using the new distance-criterion. (3) *NAR*-tree*, using the new distance-criterion and the original normalization (normalized to [0, 1] in each dimension). (4) *FER*-tree*: Fully Enhanced R*-tree, enhanced by the new distance-criterion and the extended normalization.

We use LINEITEM table (in TPC-H benchmark) having sixteen attributes, of which the six attributes, SHIPDATE (date), QUANTITY (floating point data), DISCOUNT (floating point data), SHIPMODE (character string), SHIP-INSTRUCT (character string), and RETURNFLAG (character string), are selected as index attributes since they are used in the queries of the benchmark as query attributes. The total number of tuples is 600,000. We implemented the approach proposed in the work (H. V. Jagadish and Srivastava, 2000) to handle character string attributes, where the string data is stored in a file outside the index. Thus, the number of string accesses was also measured for comparison.

5.1 Effect of the New Distance-criterion

We created two indices: the original R*-tree and AR*-tree. The number of nodes and storage utilization of each tree are shown in Table 2. Total number of nodes in the original R*-tree is about twice as many as that in AR*-tree. Since the storage utilization of the original R*-tree used for common spatial data is expected to be approximately 70% (Jurgens and Lenz, 1998), 58.7% utilization (see Table 2) is a serious degradation. On the contrary, AR*-tree can achieve reasonable storage utilization even when some attributes have small cardinalities.

Table 2: R*-tree vs. AR*-tree

	R*-tree	AR*-tree
Number of non-leaf nodes	16713	15145
Number of leaf nodes	90012	69308
Total number of nodes	106725	84453
Storage utilization (%)	58.7	72.1
Number of node accesses	46319	10201
Number of string accesses	102042	22560

Performance of range query is also presented in Table 2. The query ranges of the attributes of SHIPDATE, QUANTITY, and DISCOUNT are intervals like "2002-01-01≤SHIPDATE ≤ 2002 -12 -31", where the intervals are set to 10% of data range. The other attributes, i.e., RETURNFLAG, SHIPINSTRUCT, and SHIPMODE, are used in "equality" predicates because of their data types. The query is executed 30 times and each execution is done with randomly selected query range. Table 2 shows the search performance using the AR*-tree is much better than that using the original R*-tree.

5.2 Effect of the Extended Normalization

Using the above-mentioned TPC-H table, comparison among the three indices of AR*-tree, NAR*-tree, and FER*-tree is made in terms of group performance of range queries. Certainly, we can not exhaust all the possible patterns of range query group using six index attributes. We select range queries using three or two index attributes. Two groups of range queries are tested as examples.

Table 3 shows the attributes and their corresponding dimensions. Tables 4 and 5 are the two groups of queries.

Table 4 is Query group A having five range queries, each of which has three or two query attributes. Table 5 shows Query group B consisting of four range queries. Unlike Query Group A, each query has same number of query attributes in Query Group B. The importance degree given to each attribute, which is used to construct FER*-tree, is based on the occurrence of each attribute in the query group. Predicates concerning SHIPDATE, DISCOUNT, and QUANTITY use intervals like "$l \leq A \leq u$". Constants l and u are selected so that the selectivity of each predicate is 10%. Attributes of RETURNFLAG, SHIPINSTRUCT, and SHIPMODE are used in "equality" predicates. Cardinalities of these 3 attributes are 3, 4, and 7, respectively. Each query is executed 30 times and the average numbers of index node accesses are presented in Tables 6 and 7.

In the two tables, the numbers in parentheses are the number of strings accesses. The last row of each table indicates the total number of index node accesses of the query group. FER*-tree clearly outperforms the other two trees in terms of both total number of index node accesses and that of strings accesses. However, the AR*-tree is the most efficient when *query-1* in Query Group A and *query-3, 4* in Query Group B are concerned, which seems because that the queries using SHIPDATE and QUANITITY restrict most the nodes to be searched. These experiments indicate that the group performance of range queries is improved using the FER*-tree more than using NAR*-tree.

Table 3: Attributes and their dimensions

Attributes	Dimensions
SHIPDATE	A_1
QUANTITY	A_2
SHIPMODE	A_3
SHIPINSTRUCT	A_4
DISCOUNT	A_5
RETURNFLAG	A_6

Table 4: Query group A

	A_1	A_2	A_3	A_4	A_5	A_6
query-1	○	○			○	
query-2	○		○			
query-3		○		○		○
query-4	○			○		
query-5		○	○			
Importance Degree	3	3	2	2	1	1

Table 5: Query group B

	A_1	A_2	A_3	A_4	A_5	A_6
query-1		○	○	○		
query-2	○		○		○	
query-3	○	○				○
query-4	○	○		○		
Importance Degree	3	3	2	2	1	1

Table 6: Performance of Query group A

	AR*-tree	NAR*-tree	FER*-tree
Number of nodes	84453	83446	83653
query-1	1164(0)	2565(0)	2030(0)
query-2	8106(6704)	7730(7291)	5998(5690)
query-3	8676(14143)	3122(6211)	2601(5119)
query-4	8564(7902)	8410(8367)	6242(6225)
query-5	12533(8556)	8557(8061)	7049(6616)
Total	39046(3735)	30384(29940)	23922(23650)

Table 7: Performance of Query group B

	AR*-tree	NAR*-tree	FER*-tree
query-1	11862(19474)	3593(6932)	2873(5539)
query-2	6800(5429)	2595(2376)	2666(2463)
query-3	1188(976)	3017(3007)	1770(1746)
query-4	1409(1219)	2932(2894)	1617(1602)
Total	21259(27098)	12137(15209)	8926(11350)

6 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 and is also helpful to improve the query performance on OLAP data. In this paper, we tried to enhance the R*-tree in order to evaluate range queries on OLAP data more efficiently. It is pointed out that there often exist many slender nodes when the R*-tree is used on OLAP data, which cause some problems. This paper presented two approaches. One is to control the clustering pattern of the tuples among the R*-tree leaf nodes and then to improve the group performance of range queries. The other is to introduce a distance criterion to the insert algorithm of the R*-tree. Our proposals are discussed in detail and examined by experiments.

REFERENCES

C. Chung, S. Chun, J. Lee, and S. Lee (2001). Dynamic Update Cube for Range-Sum Queries.

Proc. VLDB Intl. Conf.,

Council (1999). TPC benchmark H standard specification (decision support)".

http://www.tpc.org/tpch/

D. Papadias, N. Mamoulis, and V. Delis (1998). Algorithms for Querying by Spatial Structure.

Proc. VLDB Intl. Conf.

H. Horinokuchi, and A. Makinouchi (1999). Normalized R*-tree for Spatiotemporal Databases and Its Performance Tests.

IPSJ Journal, Vol. 40, No. 3.

H. P. Kriegel, T. Brinkhoff, and R. Schneider (1993).

Efficient Spatial Query Processing in Geographic Database Systems.

H. V. Jagadish, N.Koudas, and D. Srivastava (2000).

On Effective Multi-Dimensional Indexing for Strings. Proc. ACM SIGMOD Intl. Conf.

J. Han and M. Kamber (2001). Data Mining--Concepts and Techniques. Morgan Kaufmann press.

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.

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

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

N. Roussopoulos, Y. K and M. Roussopoulos (1997).

Cubetree: Organizaiton of and Bulk Incremental Updates on the Data Cube. Proc. ACM SIGMOD Intl. Conf.

R. Agrawal, A. Gupta, and S. Sarawagi (1997). Modeling Multidimesnional Databases. Proc. Intl. Conf. on Data Engineering (ICDE).

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.

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

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.

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