

SQR-tree: A Spatial Index Using Semi-quantized MBR Compression Scheme in R-tree*

JONGWAN KIM, SEOKJIN IM, SANG-WON KANG, CHONG-SUN HWANG
AND SANGKEUN LEE**

*Department of Computer Science and Engineering
Korea University
Seoul 136-713, Korea
E-mail: wany@korea.ac.kr*

The increase in spatial data for location-based service (LBS) in mobile computing or geographic information system (GIS) has led to more research on spatial indexing, such as R-tree. Nevertheless, few studies have attempted to improve performance by reducing the size of the index. If the minimal bounding rectangles (MBRs) that represent objects in R-tree are compressed, the index size is reduced and location-based services are provided to the user more rapidly. This study proposes a new MBR compression scheme using MBR semi-quantization (SqMBR) scheme and SQR-tree, which indexes spatial data using R-tree. Since the SqMBR scheme decreases the size of MBR keys, halves the enlargement of a quantized MBR (QMBR), and increases node utilization, it improves the overall search performance. This scheme decreases quantized space more than existing quantization schemes. The SqMBR scheme increases the utilization of disk allocation units. In spatial index, a greater number of node entries lowers tree heights and decreases the number of node accesses, thereby shrinking disk input/output. This study analyzes the number of node accesses mathematically and evaluates the performance of SQR-tree using real location data. The results show that the proposed index performs better than existing MBR compression schemes.

Keywords: spatial index, R-tree, minimum bounding rectangle, index compression, index packing, quantization, location-based services

1. INTRODUCTION

Location is an intuitive, but important class of location-based service. Advances in mobile devices and wireless communication technologies have enabled LBS that delivers location information to mobile users. The typical spatial data management of LBS system is depicted in Fig. 1 [1], and examples of such services include finding a hospital or hotel, obtaining local maps, or acting as a tour guide [2]. Client queries are sent to a server, which searches target objects in a spatial database and returns the search results to the mobile clients. Currently, we are interested in the server-side index and query processing of location information that consists of two-dimensional (2-D) positions, such as hospital or hotel locations.

The LBS server stores an index in a spatial database to process queries, such as R-tree. To manage spatial objects, a spatial database system needs a server with considerable memory and extended processing times. To process spatial queries more effectively,

Received August 7, 2006; revised October 11, 2006; accepted November 22, 2006.

Communicated by Makoto Takizawa

* This work was supported by the Second Brain Korea 21 Project. ** Corresponding author.

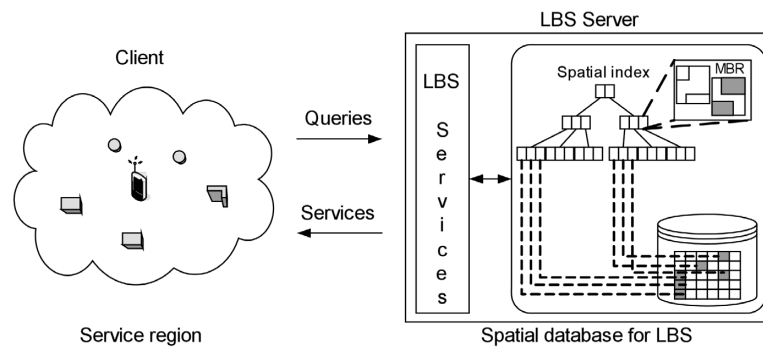


Fig. 1. LBS system architecture and spatial data management.

appropriate spatial indexes and query processing schemes are required. Existing spatial database systems process tasks, such as query, insertion, and deletion, using established index. However, a disk-based database system is limited in terms of index size, which affects search performance. In particular, when the disk allocation unit size in the operating system is fixed, then if the index is large, more blocks have to be read, and the throughput consequently decreases. If the key size is small, a block can contain more keys and fewer blocks have to be read.

For spatial indexes, especially for R-tree managing 2-D data, the minimal bounding rectangle (MBR) keys for each dimension account for approximately 80% of the index [3]. Therefore, if the keys are compressed, the number of entries stored in one node increases and search performance improves accordingly. As the entries in a node increase, the height of the index is reduced, decreasing disk I/O. Using this method, the index saves space. As spatial data have increased over the last few years, studies of spatial indexes have progressed, especially for R-tree-based indexes. Nevertheless, few studies have attempted to improve performance by reducing the size of the index.

The basic concept of this paper is to compress MBR in the spatial index. Various MBR compression schemes have been examined [3-5], but it is the quantized MBR (QMBR) scheme that produces the smallest MBR key size. The QMBR can minimize key size because it quantizes a region into n pieces and replaces the two points of MBR with quantization units. However, since real MBR has to be expanded to the quantization unit in QMBR, MBRs overlap with one another. This lowers the search performance in R-tree because of backtracking, which means that the parent node has to be accessed repeatedly in order to search sibling nodes. The low search performance in LBS server hinders the prompt return of spatial data requested by users. To overcome this problem, we propose a new MBR compression scheme that enables faster data transmission by reducing the size of coordinates and the overlap of MBR. That is, in contrast to the QMBR scheme, which expands the four sides of MBR, the proposed scheme compresses the data by using semi-quantization MBR (SqMBR) that expand only two sides of MBR.

In SqMBR scheme, coordinates are stored as follows. First, the lower-left corner of SqMBR is stored as coordinates relative to the starting point of the entire region. Since relative coordinates are offsets from the starting point, the point can be stored as values that are much smaller than the real coordinates. For the upper-right corner, the right side of MBR is aligned to the x -axis quantization unit and the top side to the y -axis

quantization unit and the units are stored as x - and y -coordinates, respectively. Compared to QMBR, Sq MBR is advantageous in that overlap is reduced as only two sides are expanded. As a result, the Sq MBR reduces server-side search time through compressing MBR by minimizing the size and overlap of coordinates. Thus, it shortens the query reply time and returns position information on locations (*e.g.*, hospitals, hotels, and taxis) quickly to the user.

In this paper, we propose a novel spatial indexing scheme and structure that process geographical data. That is, we introduce a Sq MBR scheme, and propose a semi-quantization R-tree (SQR-tree) that indexes spatial data. The Sq MBR and SQR-tree ideas are derived from our previous study, which compressed MBR, and we expanded the study with mathematical analysis and extensive experiments [6]. To the best of our knowledge, this is the first attempt to utilize semi-quantized MBR compression, which decreases the enlarged area of QMBR by 50% in two dimensions.

By decreasing the size of MBR keys, Sq MBR halves QMBR enlargement, increases node utilization in an index, and improves overall search performance. The SQR-tree increases the number of node entries by compressing MBR keys. The greater number of node entries lowers tree heights, thereby reducing the number of node accesses when processing queries, which leads to faster query results by decreasing disk I/O. The index structure is constructed by including MBR for sub-node details at each node and storing compressed MBR information in the entries. We analyzed the performance of SQR-tree by analyzing the number of node accesses in 2-D space mathematically and using real location data. The results of the experiment show that the proposed index performs better than existing MBR compression methods.

This paper is organized as follows. Section 2 discusses MBR compression schemes, section 3 presents the proposed Sq MBR scheme and some definitions for semi-quantization, section 4 implements Sq MBR and SQR-tree algorithms, section 5 discusses the improved performance of SQR-tree using location data, and section 6 summarizes the study and proposes new directions for future research.

2. MBR COMPRESSION SCHEMES

Spatial objects can be represented using coordinates, which are expressed an MBR that consist of the lower-left and upper-right coordinates (Fig. 2) [7]. Since these keys take up most of the index structure, reducing the key size by compression enables more entries to be stored in a node. Several studies have examined how to reduce the size of indexes using MBR compression scheme. Examples include relative representation of MBR (RMBR), hybrid representation of MBR (HMBR) [3], quantized representation of MBR (QMBR) [4], and a virtual bounding rectangle (VBR) [5]. The point of these studies is that they reduce MBR key size. The first two schemes, namely RMBR and HMBR, calculate the offset of a relevant MBR from a specific coordinate of the search region. By contrast, QMBR and VBR utilize quantization, which divides the search region using a grid based on a fixed value. QMBR-related methods increase the size of MBR to that of the quantized area, enlarging the search region and causing MBR to overlap. This increases the number of node accesses, decreasing overall search performance.

RMBR compresses keys by calculating the relative offset for MBR stored in each entry with reference to MBR of one node. Typically, a real MBR has absolute coordinates

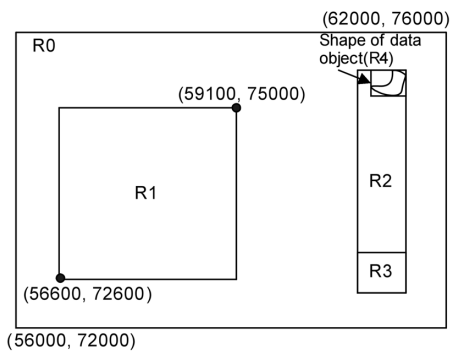


Fig. 2. Minimal bounding rectangles of objects.

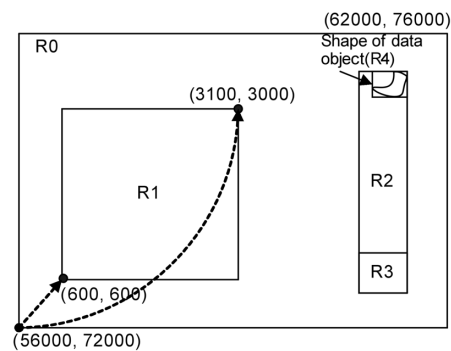


Fig. 3. Relative representation of MBR.

in a region. An MBR coordinate is stored in 4 bytes of integer type, so each MBR requires $4 \times 4 = 16$ bytes, as in R0 in Fig. 2. By using a relative offset to store the MBR key of each entry contained in a node, 8 bytes of space can be saved. In Fig. 3, if the end coordinates of R1 are calculated using the relative offset from the starting coordinate, the storage space of a coordinate is reduced to 2 bytes, further improving node utilization. For example, the end coordinates are reduced from (59100, 75000) to (3100, 3000). Since the size of the integer type is 4 bytes ($-2,147,483,648 \sim 2,147,483,647$) and a short integer is 2 bytes ($-32,768 \sim 32,767$) in programming language, x-coordinates 59,000 and 3000 need 4 bytes and 2 bytes, respectively. Therefore, 8 bytes are sufficient for RMBR. However, a shortcoming of RMBR is that if two points are far from the starting point of R0 and close to the end, the relative coordinates are large and a coordinate requires 4 bytes of storage. For this reason, the RMBR key size equals that of real MBR, and as a result, MBR compression effect doesn't occur.

The MBR key size becomes even smaller in HMBR. The rationale of HMBR is simple. In Fig. 4, the lower-left corner of HMBR is identical to that of RMBR, but the upper-right corner is calculated by the relative coordinates, height and width, to the starting point of the same MBR. As a result, the coordinate size is reduced from (3100, 3000) to (2500, 2400). The HMBR makes the end coordinates of MBR smaller so that each of them can be stored in 1 byte [3]. Thus, in order to store MBR, HMBR needs 6 bytes (left point (2, 2) + right point (1, 1)). However, HMBR has two shortcomings. First, since the relative distance from the starting point is used as in RMBR, if the starting point is far from the start of the range, the number of bytes of the key increases. Second, if the distance between the two points composing an MBR is large, the endpoint of HMBR may require 2 or more bytes. When both of these shortcomings exist, HMBR needs at least 6 bytes of storage space, which lowers performance.

The QMBR stores the coordinates of real MBR in 1 byte each, increasing the utilization of storage space. Dividing the search space by a constant integer and compressing the keys by using the space in n -number quantization can save more space than RMBR or HMBR; this scheme is referred to as QMBR. We can limit an MBR to a very small value if we replace keys with the number of quantization units by enlarging MBR in the directions of the x - and y -axes so that it can correspond to the quantization level q . Fig. 5 shows a case in which the x - and y -axes are quantized by 16×11 on each axis. If the

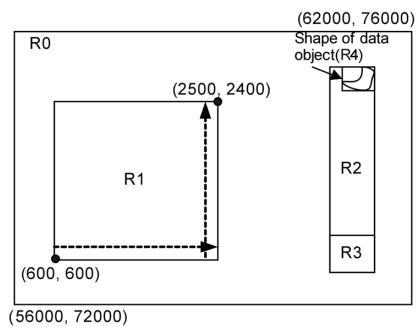


Fig. 4. Hybrid representation of MBR.

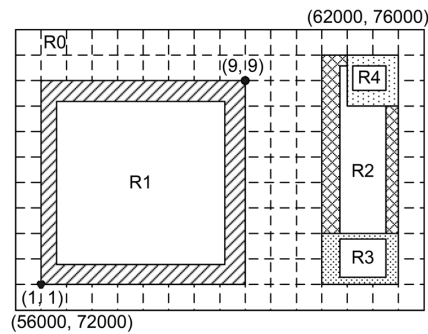


Fig. 5. Quantized representation of MBR.

quantization level is smaller than 256, each coordinate is stored in 1 byte. The concept of VBR, proposed in A-tree [5], reduces the number of units required to store keys by quantizing the search region, as with QMBR. In other words, it constructs a VBR by enlarging MBR to the closest quantization unit. Unfortunately, these QMBR variants make MBR larger, which consequently enlarges the search region, causing the relevant region to be larger than real MBR, when searching for objects. As a result, overlap with other MBR occurs, and the number of node accesses increases, degrading the overall search performance.

Although the results in Figs. 3-5 show that the keys are stored after compression, RMBR and HMBR schemes need 8 and 6 bytes of storage space, respectively. With the QMBR scheme, overlapping due to enlargement of MBR occurs, affecting search performance. RMBR and HMBR enable keys to be stored in fewer bytes by calculating the relative offset from the starting coordinates of the search region. In RMBR, a key value stored in 16 bytes is reduced to 8 bytes, while it is further reduced to 6 bytes in HMBR. Nevertheless, HMBR has a disadvantage that key storage requires 2 bytes more than in QMBR. As discussed above, existing compression methods have several shortcomings that *Sq*MBR method proposed in this paper resolves. The details are explained in section 3. Table 1 summarizes the compression schemes.

Table 1. Summary of MBR compression schemes.

Scheme	Approximation	Size	Description
MBR	Two points of a rectangle	16	No compression
RMBR	Relative coordinates	8	MBR size/2 bytes
HMBR	Endpoint = (height, width)	6	Start point = RMBR
QMBR	Quantization of a space	4	MBR enlarges

3. SEMI-QUANTIZING MBRs

In this section, we describe *Sq*MBR scheme. First, we provide three definitions and a lemma for the representation of *Sq*MBR. Next, we analyze *Sq*MBR size according to the number of cells occupied by MBR within the quantized range, and show that *Sq*MBR scheme expands space much less than the existing QMBR scheme.

3.1 Representation of Semi-quantization MBR

Since an n -dimensional rectangle, such as MBR, can be viewed as a $2n$ -dimensional point [8], point compression saves index space and increases search performance. Four points represent a 2-D rectangle. In semi-quantization MBR scheme, each node has an MBR that comprises all entries stored in that node. The MBR is represented by two endpoints (α, β) , where $\alpha = (\alpha.x, \alpha.y)$ and $\beta = (\beta.x, \beta.y)$. The aim of SqMBR scheme is to represent α as a relative value, and to quantize β in order to halve the size of the expanded area, a region without objects due to the expanded MBR, in QMBR. This also minimizes the storage space required for keys and improves search performance. We present three definitions of semi-quantization.

Definition 1 *Representation of the relative coordinates of MBR*

Let M be the MBR of the entire search space; then, the lower-left and upper-right corners of M are represented by $(M.lx, M.ly)$ and $(M.rx, M.ry)$, respectively. An entry has two points (α, β) , and the relative coordinate for starting point α is as follows:

$$R_M(\alpha) = (|M.lx - \alpha.x|, |M.ly - \alpha.y|). \quad (1)$$

Definition 2 *Quantization of MBR*

Let (M_s, M_e) be the two points of M using Definition 1, and q be the quantized level; then, quantized β is defined as follows, where Q_e is the endpoint of a quantized MBR:

$$Q_e(\beta) = \begin{cases} 1, & (\beta = M_s) \\ \lceil ((\beta - M_s)/(M_e - M_s)) \times q \rceil, & \text{otherwise.} \end{cases} \quad (2)$$

Endpoint β is transformed into Q_e according to the quantized level, which minimizes the storage space required in bits. The quantized levels, 2^n ($n = 0, 1, 2, \dots, 8$), are represented by 1 byte. Since Q_e is determined by the quantized level q , it can be represented using a bit string. In other words, the binary representation is $(Q_e)_2$ and the length of the bit string is $\log_2 q$. The endpoint is stored as $(Q_e - 1)_2$. For example, if $Q_e(\beta.x) = 9$ and $Q_e(\beta.y) = 9$, the bit string is 10001000, the concatenation of the two binary codes in Fig. 6. As a result, R1 requires only 5 bytes of storage.

Definition 3 *False-overlap region (FOR)*

Let *quantized_MBR* be the enlarged region in quantization; then, the search region is enlarged by expanding an MBR to the closest quantization unit. If the enlarged region is defined as *FOR*, which causes MBR to overlap, it is as follows:

$$FOR = \text{Quantized_MBR} - \text{MBR}, \text{ when } (\text{MBR} < \text{Quantized_MBR}). \quad (3)$$

In QMBR, the FOR area appears in the four sides of MBR as in Fig. 5, but in SqMBR it appears only in the upper and right sides of MBR as in Fig. 6. The region is half that in QMBR scheme. The coordinate space also decreases to a minimum of 5 bytes.

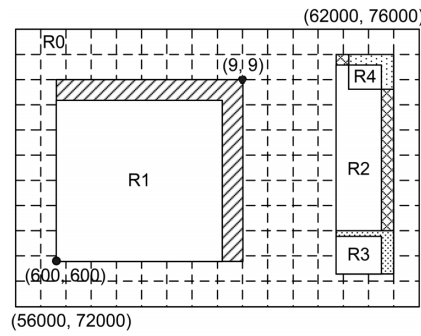


Fig. 6. Semi-quantized MBR.

Although high levels of quantization increase the number of bits, thereby increasing the stored bytes of keys, the *Sq*MBR scheme still maintains an advantage over other methods.

3.2 Scaling *Sq*MBR Size in Quantized Areas

We now present an analysis of MBR overlap size to show that the quantized area of *Sq*MBR is smaller than that of QMBR. The QMBR size is calculated by counting the number of cells overlapping MBR. The expansion of real MBR that results from quantization is determined using the relationship between MBRs in a specific search space $[0, 1]^2$ and quantized cells. We assume that MBR and quantized cell are square, because square is more straightforward than rectangle when calculating the number of cells occupied by an MBR, as in Fig. 7.

Each quantized cell is divided into four parts according to the position of the lower-left corner α of MBR, and the number of cells overlapping MBR differs. When G (grid) is the quantized area, c is the side of each cell, and n is the number of overlapping cells, and then MBR side r can be presented as follows:

$$r = n \times c + \Delta x, n = \left\lfloor \frac{r}{c} \right\rfloor, \Delta x = r - n \times c. \quad (4)$$

The cells overlapped by the MBR in a domain are illustrated in Fig. 7; the number of cells differs depending on where α is located (g_1 , g_2 , g_3 , and g_4 are the four parts of each cell). Therefore, the average area occupied as a result of quantization can be calculated by determining the average overlap size in the corresponding domain using the probability that MBR belongs to each area. To calculate the probability, we need to determine the area of each part. As the rectangle belonging to g_1 overlaps four cells, its area is $(c - \Delta x)^2$. The size of g_2 and g_3 , which overlap six cells, is $2\Delta x(c - \Delta x)$ in Fig. 8. The size of g_4 equals Δx^2 and overlaps nine cells (Fig. 9). Using this information, we can obtain the average size of the area occupied by an MBR in a domain. That is, we can calculate the size occupied by MBRs in the quantized area by counting the average number of cells overlapping an MBR. With respect to cells, we can divide the size of an MBR into three cases:

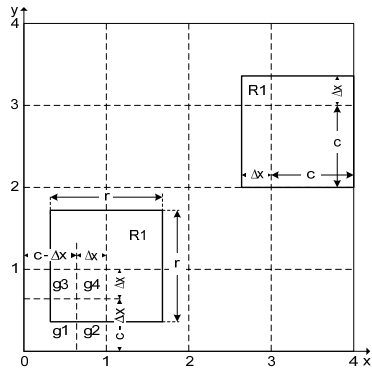


Fig. 7. The number of cells in g1.

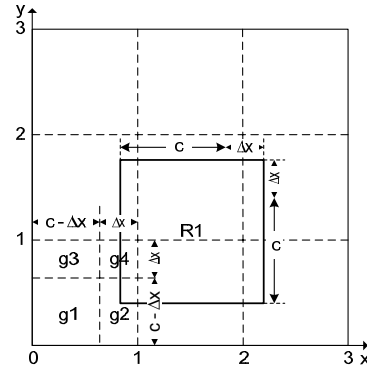


Fig. 8. The number of cells in g2 and g3.

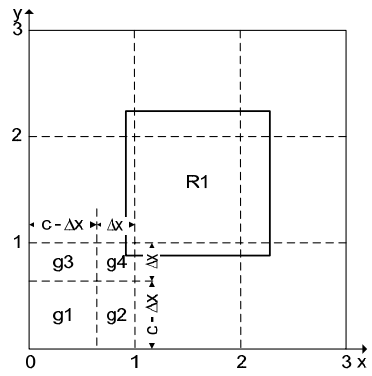


Fig. 9. The number of cells in g4.

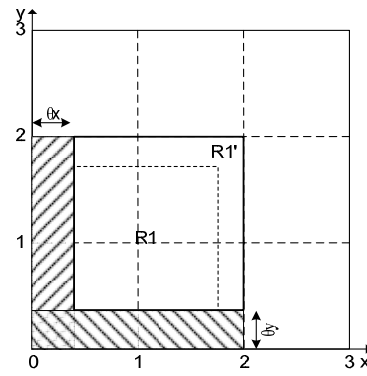


Fig. 10. Unexpanded areas of SqMBR.

Case 1: The length of an MBR side is longer than that of a cell ($r > c$).

The number of cells overlapping MBR is $n \times n$ when α of MBR is in g1, as in Fig. 7, or $n(n + 1)$ when α is in g2, as in Fig. 8. The same number of cells is occupied when α is in g3 because areas g2 and g3 are the same size. Lastly, the number of cells is $(n + 1)(n + 1)$ when α of MBR is in g4, as in Fig. 9. If the distribution is assumed to be even, the probability that a corner of MBR is in g1, g2, g3, or g4 is $(c - \Delta x)^2/c^2$, $2\Delta x(c - \Delta x)/c^2$, $2\Delta x(c - \Delta x)/c^2$, or $\Delta x^2/c^2$, respectively. Accordingly, the average number of cells overlapping MBR is as follows, where the average area occupied by m MBR is mr^2 :

$$AveCells(r > c) = \frac{n^2(c - \Delta x)^2 + 2n\Delta x(n + 1)(c - \Delta x) + \Delta x(n + 1)^2}{c^2} = \frac{(nc + \Delta x)^2}{c^2}. \quad (5)$$

Case 2: The length of an MBR side is shorter than that of a cell ($r < c$).

The numbers of cells when one side of an MBR meets g1, g2 (g3), and g4, are 1, 2, and 4, respectively, and the probability is the same. Therefore, the average number of cells is as follows, and the average area of m MBR is $m(c + \Delta x)^2$:

$$\begin{aligned}
AveCells(r < c) &= \frac{1 \times (c - \Delta x)^2 + 2 \times 2\Delta x(c - \Delta x) + 4 \times \Delta x^2}{c^2} \\
&= \frac{c^2 + 2c\Delta x + \Delta x^2}{c^2} = \frac{(c + \Delta x)^2}{c^2}.
\end{aligned} \tag{6}$$

Case 3: The length of an MBR side equals that of a cell ($r = c$).

In this case, a corner meets four cells wherever it is positioned, so the average area of MBR is $4c^2$.

In quantization, MBR expands vertically and horizontally, while in semi-quantization, unexpanded areas, θ_x , θ_y , exist, as in Fig. 10. The reason that unexpanded areas exist is that only the right and top sides of R1 expand to quantization units. In Fig. 10, R1' is a semi-quantized MBR, the expansion area of which is smaller than that of QMBR. If QMBR scheme is applied to R1, the bottom side and the left side of R1 meet the x - and y - axes, respectively. As a result, θ_x and θ_y , which did not expand in SqMBR are also enlarged. When θ_x and θ_y denote the unexpanded areas in the x - and y -axis of R1, respectively, the unexpanded area (\bar{Q}) of MBR is as follows:

$$\bar{Q} = 4nc(\theta_x + \theta_y) + \theta_x\theta_y. \tag{7}$$

Accordingly, the final formula for the average size of cells expanded in semi-quantization is as follows. We assume that quantization and MBR involve square, but MBR can be rectangular and the formula can be applied in a straightforward manner for such a case:

$$CellSizeSQ = \begin{cases} r^2 - \bar{Q}, & \text{if } (r > c) \\ 4c^2 - \bar{Q}, & \text{if } (r = c) \\ (c + \Delta x)^2 - \bar{Q}, & \text{otherwise.} \end{cases} \tag{8}$$

When an MBR is expanded to the closest quantization unit, if the actual position is coincident with the quantization unit, the corresponding side does not need to be expanded. In the existing quantization scheme, if two MBR sides are coincident with the quantization unit, the other sides are expanded. Moreover, the sides of MBR do not expand if all four sides of MBR are coincident with the quantization unit. By contrast, in semi-quantization, expansion does not occur even if only two sides are coincident. This is because the two sides forming the lower-left corner of MBR are calculated using their location relative to the entire area. Reflecting this feature in the algorithm, we can save CPU cost and provide faster location-based services.

Lemma Let α and β be the two points of an MBR. For any MBR k and quantized line l , it holds that if a side of real MBR coincides with the quantization line, the corresponding side does not expand.

Proof: To prove this lemma, it can be converted into the contrapositive, which holds that if one side of MBR is expanded, it then differs from the quantization line. By definition, if one side of MBR is expanded to the quantization line, it does not coincide with the

current quantization unit, and the opposite is true. Accordingly, the sides of MBR that are not coincident with the quantization line are expanded. The properties of the expanded point β are as follows:

$$\left\lfloor \frac{\beta.x}{q} \right\rfloor < \beta.x < \left\lceil \frac{\beta.x}{q} \right\rceil, \quad \text{mod}(\beta.x, q) \neq 0. \quad (9)$$

$$\left\lfloor \frac{\beta.y}{q} \right\rfloor < \beta.y < \left\lceil \frac{\beta.y}{q} \right\rceil, \quad \text{mod}(\beta.y, q) \neq 0. \quad (10)$$

Therefore, the endpoint of MBR β does not coincide with the quantization line of the x - and y - axes and expansion occurs. \square

Correctness: If the coordinate of real MBR is coincident with a multiple of the quantization level, the corresponding side coincides with one of the quantization lines. In this case, the size of MBR is maintained even after quantization, and the number of node accesses does not increase because no expansion occurs.

As presented above, the area in the proposed scheme is smaller than that in existing quantization schemes, and each side of MBR is coincident with the quantization unit; consequently, the volume of calculation decreases. As regards the effect on search performance of an expansion of MBR, section 5 contains a mathematical analysis that shows that the result of the experiment is identical.

4. IMPLEMENTATION OF SEMI-QUANTIZATION R-TREE

SQR-tree is a height-balanced tree based on R-tree. The differences between SQR-tree and R-tree are insertion and searching. In this section, we discuss the index structure and SQR-tree algorithm.

4.1 Index Structure

In Fig. 11, R0 represents the search region, and the areas R1–R7 contain MBR information (solid lines). Each MBR is quantized as it expands along the x - and y -axes. Since R2 is adjacent to the boundary of the parent MBR, the coordinate of its expanded area is the maximum value of its quantized level. If the sides to be expanded coincide with the quantization unit, as in R2, MBR does not expand, as proven in *lemma* R3, R4, and R5 expand into the area of R0 and R1 (dotted lines). The expanded areas in the upper and right regions of R6 and R7 represent the endpoints.

The MBRs from R0 to R7 are included in SQR-tree with additional attributes. Fig. 12 is an example. The SQR-tree comprises MBR based on the minimum approximation of the objects that represents the entire region of node entries, as well as a pair of (*child_ptr*, *Sq*(MBR)) entries with information on sub-node pointers and the expanded MBR. As shown in Fig. 13, a node has up to a maximum of m entries, and a *flag* that distinguishes whether the node is a leaf or an internal node.

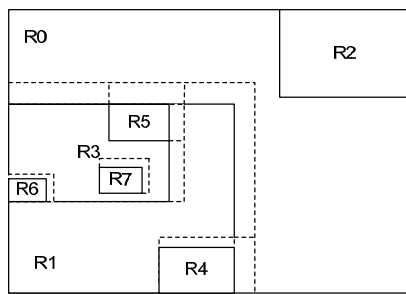


Fig. 11. Search space and MBR.

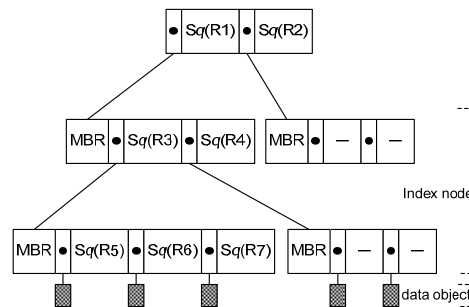


Fig. 12. Index structure of SQR-tree.

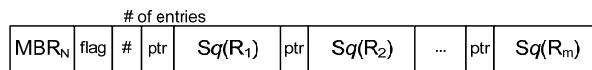


Fig. 13. Node structure.

The root node calculates the $Sq(MBR)$ of the entire space, and does not possess information on the node MBR. The real MBR of an entry is calculated from $Sq(MBR)$, parent MBR, and sub-MBR. That is, the child $Sq(MBR)$ in a node can be calculated from the parent MBR in the same node and the child MBR. Accurate information on each entry is used to prune the nodes.

4.2 SqMBR Conversion and Insertion Algorithm

Since SQR-tree algorithm is based on R-tree, this section discusses only the differences between the two trees. The major differences concern insertion and searching. To insert an object, SQR-tree searches down from the root node to the leaf node, calculates $SqMBR$ of the object, and compares it to the entry for insertion.

In the detailed explanation of the algorithm, in the procedure, M indicates the entire region and E denotes the object to be inserted. To visit the root node, the index is loaded from the disk to memory (line 4). At that time, semi-quantization is performed only when the entry E that is to be inserted in the index is smaller than M , and E is inserted in SQR-tree. The starting and end coordinates of $SqMBR$ are converted using Eqs. (1) and (2), and are stored in entry E (lines 5-16). The converted E is designated a data candidate (line 17), and inserted data candidates are added to the index (lines 19-30).

Algorithm *SqMBR conversion and insertion: Insert a new index entry E into SQR-tree*
 Procedure SQRTree_Insert (Region M , Entry* E)
Input: a search region M , an object entry pointer E
 Begin
 1: *data_candidates* is a pointer variable of the *LinkList** type;
 2: *root_ptr* is a pointer variable of the *RTNode** type;
 3: *entry_data* is a variable of *Entry* type;
 4: Load root node into memory;
 5: **if** a search region M is greater than entry E ;
 6: **then** /* calculate the start point of $SqMBR$ */

```

7:   Calculate the left-corner x-coordinate  $lx$  by  $abs(M.lx - E.lx)$ ;
8:   Calculate the left-corner y-coordinate  $ly$  by  $abs(M.ly - E.ly)$ ;
9:   Update the start point of an entry  $E$  by the relative distance;
10: else return - 1;
12: if the  $E$ 's endpoint is equal to the  $M$ 's start point;
13: then /* calculate the endpoint of SqMBR */
14:   Set the  $E$ 's endpoint as 1;
15: else /*  $q$ : quantization level */
16:   Calculate and set the  $E$ 's endpoint by  $Ceiling(q * (E.ep - M.sp)/(M.ep - M.sp))$ ;
17: Insert  $E$  into  $data\_candidates$  as the first entry;
18: Delete the entry  $E$  after insertion;
19: Do repeat, if an entry exists in  $data\_candidates$ ;
20:   if  $data\_candidates$  is not null, that is, an entry exists
21:     then
22:       Construct a new entry  $entry\_data$  with dimension and null object;
23:       Set a  $data\_candidates$  to  $entry\_data$ ;
24:       Erase  $data\_candidates$ ;
25:       Insert an entry  $entry\_data$  into root, recursively;
26:     else report an error message; return - 1;
27:   if root overflows
28:     then
29:       Construct a new entry  $de$  with ( $dimension, root\ pointer$ );
30:       Split a root node from  $de$  in R-tree;
30: End Do;
End

```

However, if the number of entries in a node reaches the maximum and a new entry cannot be inserted, the R-tree split algorithm is applied (line 29). The value q is a constant for the quantization level. Another difference is that SQR-tree compares the quantized endpoint of query region Q with SqMBR key of each entry. Quantization is processed using Definition 2 (lines 5-16). An advantage of doing so is that the two coordinates, the query region Q and an entry, can be compared even though SqMBR is not restored to the original coordinate.

5. PERFORMANCE EVALUATION

In an index, search performance can be improved by increasing node size or compressing MBR keys. In this section, we analyze the number of node accesses mathematically, and evaluate the performance of SQR-tree.

5.1 Structural Analysis of the Index

Compression of index keys improves node utilization, and affects the height of the tree. When node utilization is the same, then if the index is compressed, the number of entries in a node increases. In this study, we can see that if the node utilization of each compared scheme is changed, the number of entries stored in the node increases con-

tinuously. In Fig. 14, when the utilization is 80%, the number of entries doubles in MBR and RMBR, and triples in QMBR, which uses quantization. Here, the size of a node is fixed at 4096 KB, and the maximum number of entries in a node is not set because the size of the node is proportional to its utilization. If the node size is changed, the number of entries in a node increases. At that time, if keys are compressed, the number of entries increases considerably, as in Fig. 14, and more keys are stored. Consequently, QMBR contains more entries than *Sq*MBR. That is, when the index is compressed, the number of entries stored in a node is larger in QMBR. However, this simply indicates an increase in the number of entries in relation to node size and utilization, while the actual performance is the opposite. The performance of QMBR is poorer than that of *Sq*MBR because the overlap of each MBR increases due to FOR resulting from quantization in QMBR. FOR is discussed in sections 5.2 and 5.3. When the node utilization is 70 and 80%, the index size of each scheme is as in Fig. 15. Here, the *Sq*MBR index is also slightly larger than that of QMBR.

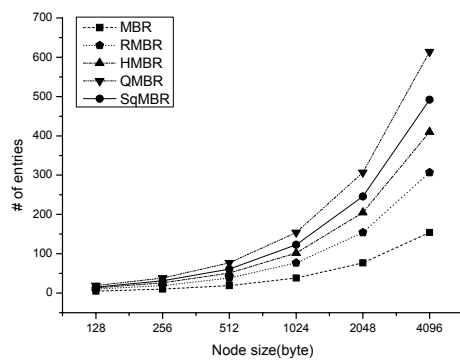


Fig. 14. Number of entries with node size.

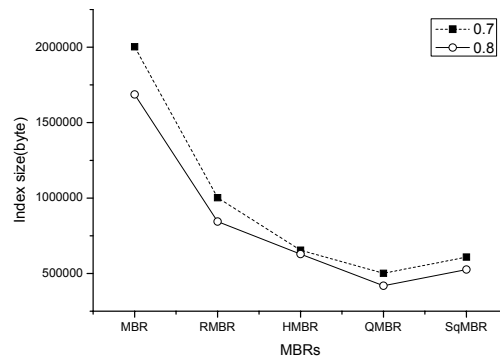


Fig. 15. Index sizes according to utilization.

In conclusion, when the index structure is analyzed without considering the search procedure, schemes using quantization store more entries and form a very small index in comparison with other schemes. If the fan-out increases, the index height is lowered, but not by much, because the change is on a log scale. However, if the level is lowered, it affects the number of node accesses and search time, and this is easily proven experimentally.

5.2 Analysis of the Number of Node Accesses

The expanded area of *Sq*MBR becomes much smaller than that of QMBR as analyzed in section 3.2. The reduction minimizes overlapping resulting from the expansion of MBR and guarantees fast results from SQR-tree. To explain this mathematically, this section provides formulas for calculating the number of node accesses in consideration of dimensions and expands them to *Sq*MBR. In this analysis, we assume for the sake of simplicity that data objects are uniformly distributed in the domain, and that MBR is square. However, we use skewed real dataset in order to reflect the realities of practical experiments in section 5.3.

The mathematical analysis of the number of node accesses in R-tree is outlined in [4] and we modify the equations. All nodes are assumed to have MBRs of equal height. Let h denote height and M_h denote the number of nodes of height h . Then, M_h equals the result of the ceiling in Eq. (11). Defining the average region of a node as a_h in a tree with height h , a_h of each node is $1/M_h$. Using the Minkoski sum [9], the probability that a node of height h will overlap a given query region is $(\sqrt[d]{s} + \sqrt[d]{a_h})^d$, where s denotes the size of the query rectangle. The number of height- h nodes that overlap the query rectangle is $M_h (\sqrt[d]{s} + \sqrt[d]{a_h})^d$; this is represented as follows, where N is the total number of data and f is the average fan-out of the leaf nodes:

$$\left(1 + \sqrt[d]{\left\lceil \frac{N}{f^h} \right\rceil} \cdot s \right)^d. \quad (11)$$

The total number of node accesses from the root to the leaf nodes in R-tree equals the sum of the nodes at each height, as represented by Eq. (12).

$$1 + \sum_{h=1}^{\lceil \log_f N \rceil - 1} \left(1 + \sqrt[d]{\left\lceil \frac{N}{f^h} \right\rceil} \cdot s \right)^d \quad (12)$$

When the quantized level q is applied, each node has a quantized cell of q^d . Since access to the nodes in QMBR scheme is first conducted at nodes of height h , followed by the sub-nodes, the probability is $(\sqrt[d]{s} + \sqrt[d]{a_h/q} + \sqrt[d]{a_{h-1}} + \sqrt[d]{a_h/q})^d$. Since this applies to all of the nodes from the root to the leaf nodes, the total number of node accesses is as shown in Eq. (13). Here, QMBR scheme accesses more nodes than MBR scheme because QMBR is bigger than real MBR owing to quantization. In the experiments, however, since QMBR maintains coordinates of smaller size than MBR, the number of node entries increases, which in turn reduces the number of node accesses.

$$1 + \sum_{h=1}^{\lceil \log_f N \rceil - 1} \left(1 + \sqrt[d]{\left\lceil \frac{N}{f^h} \right\rceil} \cdot s + \sqrt[d]{\left\lceil \frac{N}{f^{h+1}} \right\rceil} \cdot s/q \right)^d \quad (13)$$

Eq. (13) denotes the expanded sides of MBR with quantization, and is modified into Eq. (14) to reduce the expansion by half. When MBR is enlarged to the quantization region, it includes FOR area, as in Definition 3. By contrast, in SqMBR, enlargement does not take place at the starting point of MBR, so FOR is reduced by half.

$$1 + \sum_{h=1}^{\lceil \log_f N \rceil - 1} \left(\left(1 + \sqrt[d]{\left\lceil \frac{N}{f^h} \right\rceil} \cdot s + \sqrt[d]{\left\lceil \frac{N}{f^{h+1}} \right\rceil} \cdot s/q \right) - (FOR/2) \right)^d \quad (14)$$

As in Eq. (14), SqMBR can maintain an expansion area only half that of the equiva-

lent QMBR, and the probability of overlap with other nodes also decreases. This feature identified in mathematical analysis is reflected by experimental analysis. That is, *Sq*MBR scheme has fewer node accesses than QMBR scheme and this is directly linked to search performance. This analysis assumes 1,000,000 objects, a query range of 0.01%, a pointer for each entry of 4 bytes, and MBR size of each entry of 16 bytes. The keys in 2-D space are set at 8, 6, 4, and 5 bytes, for RMBR, HMBR, QMBR, and *Sq*MBR, respectively. The quantization level is set at 16. The experimental results are presented in the next section.

5.3 Experiments

In this section, we describe two sets of simulation-based experiments of *Sq*MBR scheme using real dataset. The first set of experiments illustrates the number of node accesses of MBR schemes and studies the performances of several parameters. The second set of experiments focuses on the search time. To measure the practical impact of our method, we compared SQR-tree with MBR, RMBR, HMBR, and QMBR schemes.

5.3.1 Simulation setup

We list the set of parameters used in the simulation in Table 2. The major parameters in the experiment are the node size, query region, quantization level, and object batches. In all of the experiments, the parameters take their default values if not specified otherwise. This experiment used the SEQUOIA dataset from the Sequoia 2000 Global Change Research Project, which contains the location coordinates of 62,556 California giant sequoia [10] as in Fig. 16. The dataset contains the coordinates of real spatial data, and is practical because of its skewed distribution compared to the uniform data assumed in the previous analysis. The schemes were implemented and performed using Visual C++. To eliminate the influence of background processes in Windows, we applied the CSIM simulator [11]. MBR scheme was performed using R-tree, which is a 2-D index. Existing compression scheme algorithms and SQR-tree were implemented by modifying R-tree. We used a Pentium-IV 2.6-GHz CPU with 1 GB of memory, running on Windows XP Professional.

5.3.2 Number of node accesses

In Fig. 17, the performance is measured in terms of processing queries for point and range queries. The proportion of the query region in the entire search space was set at a range of 1 to 6%. We generated 10,000 different query rectangles of the same size, and averaged the results. Here, the quantization level of both QMBR and *Sq*MBR scheme was set at 256.

Fig. 17 (a) measures the average number of node accesses for point queries according to the node size in each MBR scheme. The average value was calculated by applying the query rectangle set earlier to each method 10,000 times. First, the result of real MBR scheme is the average number of node accesses from the root to the leaf node, as in Eq. (12). The result shows that the MBR has a value much larger than other methods. In Eq. (13), the QMBR seems to increase more than MBR, but as explained in the analysis, the number of node accesses is much smaller in the practical experiment. This is because

Table 2. Experimental parameters.

Parameters	Values
Node size (bytes)	512 ~ 4K
Query range (%)	1 ~ 6
Buffer size (bytes)	4K
Quantization level	0, 8, 16, 32, 64, 128, 256
Initial Fan-out	200
# of objects	10K ~ 60K
SEQUOIA dataset	62,556 location points

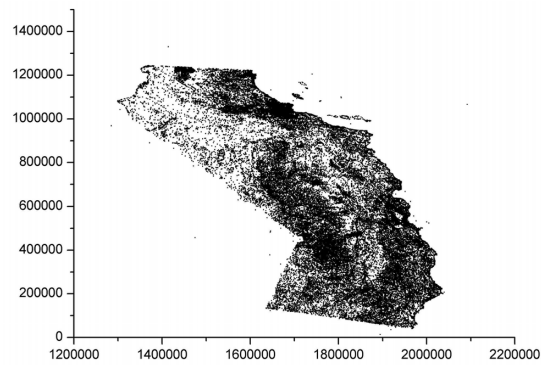
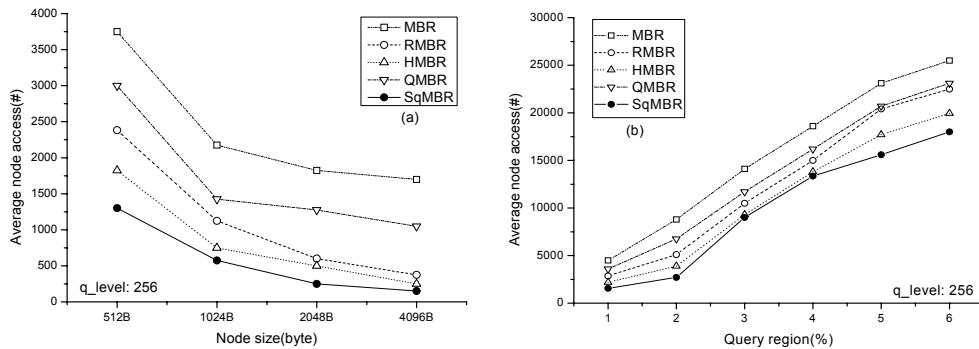


Fig. 16. Location data of California giant sequoia.



(a) With an increase in node size, bucket utilization increases due to index compression and the average number of node accesses decreases as a result.

(b) As the query region increases in size, the overlapping region of *SqMBR* decreases more than that of *QMBR*.

Fig. 17. Average number of node accesses of (a) point and (b) range query.

the number of entries stored in a node is increased by compressing key values and as a result the number of node accesses decreases. The number of accesses is smaller in RMBR and HMBR schemes than in QMBR scheme. As shown in Table 1, although the key space required in the two schemes is 8 and 6 bytes, respectively, both of which are

larger than that of QMBR scheme, QMBR scheme accesses more nodes because of overlapping MBR. This shows that the biggest shortcoming of QMBR scheme is the expansion that appears in key compression. The Sq MBR scheme reduces the expansion area, as shown in Eq. (14). Since the quantization level is 256, Sq MBR is compressed to 6 bytes. This is the same size as that in HMBR scheme, but Sq MBR scheme shows slightly higher performance because HMBR key size becomes larger than 6 bytes due to its shortcomings (see section 2). That is, the number of entries in a node decreases and the number of node accesses increases. However, Sq MBR scheme maintains 6 bytes constantly, thus overcoming the shortcomings of QMBR scheme, and improves search performance by compressing MBR into smaller key than RMBR and HMBR schemes.

Fig. 17 (b) reveals the average number of node accesses according to the size of the query region in range queries. In the experiment, the query region was set at 1 to 6% of the whole region. There were fewer node accesses for compressed MBR than for non-compressed MBR in all query regions. This is attributable to the increased fan-out of each node due to the decrease in MBR keys. Consequently, the number of node accesses also decreases. Since QMBR scheme stores one point of even the lowest level in 2 bytes, levels were set at equivalence. RMBR and HMBR schemes performed better than QMBR scheme due to the false-overlap region in QMBR. Since the increased size of nodes allows more node entries, the number of node accesses is reduced in all schemes. Here as well, due to the shortcomings of HMBR scheme, the number of node accesses is larger than that of Sq MBR scheme.

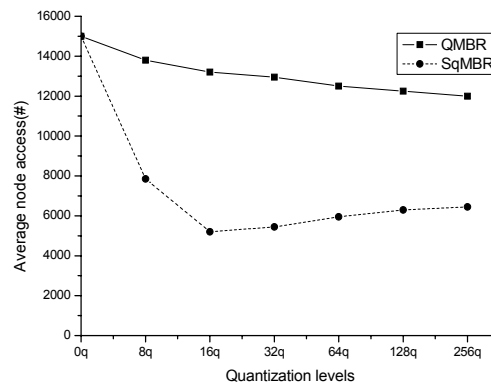


Fig. 18. The number of node accesses of range query, depending on q level. Since FOR area differs according to the degree of quantization, it also affects the number of node accesses.

Fig. 18 shows the average number of node accesses in QMBR and Sq MBR schemes, adjusting for the size of quantization. The query pattern in this experiment was based on range queries in SEQUOIA dataset, and differs from previous experiments; 10,000 query rectangles of different sizes were executed. The setting of different query rectangles was to see whether the performance of Sq MBR scheme is maintained under various conditions; as expected, the Sq MBR showed satisfactory performance.

In the experiment, since quantization is not applied to the search region if the quantization level is 0, Sq MBR scheme shows the same performance as that of MBR scheme.

At quantization level 8, where quantization is first applied, QMBR and Sq MBR schemes result in a smaller number of node accesses than at level 0. In the case of Sq MBR scheme, the number of node accesses drops significantly. In addition, quantization units are denser when the quantization level of Sq MBR is 16 than when it is 8, and as a result, even if the keys are stored in 5 bytes as in level 8, overlap among MBRs becomes much smaller. Accordingly, the number of node accesses decreases. By contrast, in QMBR scheme, the number of node accesses that decreases at level 8 does not diminish substantially even when the quantization level is raised. This is to be expected, because the area of QMBR expanded on four sides still overlaps with other MBR.

In Sq MBR scheme, the number of node accesses gradually increases from quantization 32. This occurs because quantization units become denser but the number of entries per node decreases, as in Sq MBR scheme the key is stored in 6 bytes. From quantization level 32 to level 256, coordinates are stored in 6 bytes. The reason that the Sq MBR scheme produces better results than QMBR scheme is that the overlap with other MBR is small in Sq MBR, although Sq MBR stores the key in 5 bytes at levels 8 and 16, which is larger than in QMBR. This shows that the number of entries in a node affects the number of node accesses, but the decrease in the expansion area in quantization is more closely related to performance.

5.3.3 Search time

As shown in Fig. 19, search times reflect node-access patterns. That is, search times are longest in MBR scheme, and become progressively shorter in QMBR, RMBR, HMBR, and Sq MBR schemes, in that order. This experiment used a quantization level of 256, as in the previous experiment. In Fig. 19 (a), as the node size grows, the search time of all schemes is quickly minimized. However, the decrease in search time is slow from a node size of 1024. Performance is worse using QMBR scheme than using the other compression methods owing to the increased search region. Consequently, performance is better using RMBR and HMBR schemes than QMBR scheme. Note that although the size of QMBR keys is reduced to 4 bytes, the false-overlap region that results from

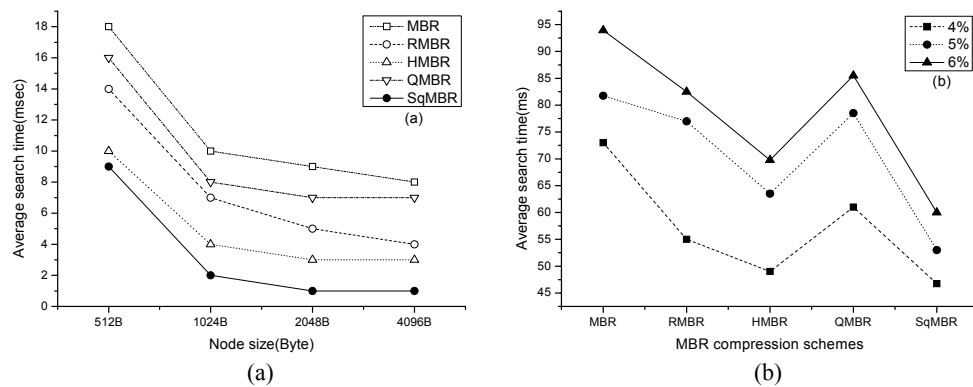


Fig. 19. Search time of (a) point and (b) range query. As the node size increases, the entities in a node increase, but the overlap of MBR also increases in QMBR scheme.

enlargement by quantization causes backtracking. Therefore, the search time increases. From the experiment involving point queries (Fig. 19 (a)) it is evident that when a larger node is applied to QMBR, the search time of QMBR will become longer than that of MBR. In addition, the search time of *Sq*MBR may increase with increasing node size, but higher performance than other schemes is maintained.

The average search time of each scheme according to the query range, 4-6%, is shown in Fig. 19 (b): the larger the query range, the longer the search time is. What should be noted here is that QMBR scheme has a longer search time and more node accesses than other methods like it. In addition, when the search range increases from 4 to 5%, search times increase more in RMBR, HMBR, and QMBR schemes than in *Sq*MBR scheme. The increased range affects search times because it increases the number of nodes. Search time is also affected by the size of MBR stored in each scheme. Particularly in QMBR, search times are affected by the increase in overlap with other MBR. By contrast, search times increase equally with the search region in *Sq*MBR because *Sq*MBR keys are smaller than those of RMBR and HMBR, and the overlap with other MBR is small in *Sq*MBR.

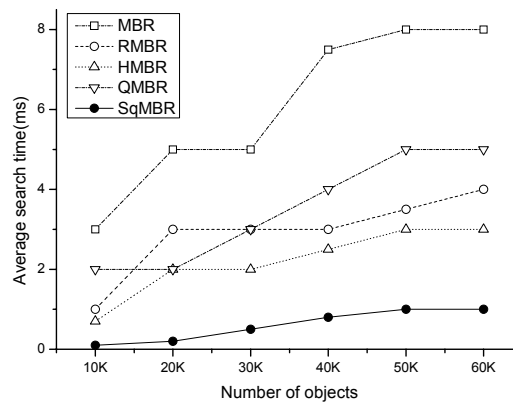


Fig. 20. Search time with object batches. Even when the amount of data increases, the performance is not markedly lowered.

Fig. 20 shows search performance according to the volume of spatial data. For the experiment, 62,556 spatial data were divided into 10K to 60K batches. The distribution of data was maintained equal to that of the real dataset, and the five MBR methods were tested using the same data distribution and data size, respectively. The quantization level was 256 for all schemes. We used 3000 point queries searching for different objects, and the average search times are presented in Fig. 20. Each scheme performs differently according to the object, but in *Sq*MBR, the search time increases slowly from 20K and above. The search time does not increase greatly, despite the increased amount of data in *Sq*MBR scheme, because coordinates are stored in a quantization unit. In *Sq*MBR scheme as well, however, search times increase with increasing data size because of overlap with other MBR.

In QMBR, with increases in data size up to 50K, search times increase proportionally. This occurs because at the same quantization level, overlap with other MBR in-

creases along with increasing data size. Search times do not increase greatly over 50K, because when overlap with other MBR exceeds a certain level it does not markedly affect search times. Here, we can see that with increasing data size, the index size and search time also increase. Moreover, in MBR compression using quantization, the increase in overlap among MBRs lowers performance. Through experiments (Fig. 20), we can see that *Sq*MBR scheme outperforms QMBR scheme under different conditions using datasets of various sizes.

In this study, we carried out experiments on MBR compression schemes seeking faster location-based services. To overcome the shortcomings of existing MBR schemes, we tested the proposed compression scheme in terms of the number of node accesses and search time. We confirmed that *Sq*MBR scheme has fewer node accesses and shorter search times than existing compression methods such as RMBR, HMBR, and QMBR schemes.

The experimental results are summarized as follows. If node size increased in the index, the number of entries in a node increased; this reduced the number of node accesses and search times in each scheme for both point queries and range queries. Furthermore, if MBR was compressed, the number of entries stored in a node increased further, and as a result, compression schemes showed much higher performance than real MBR. In particular, the performance of *Sq*MBR appeared higher than that of QMBR, which use the quantization technique, suggesting that in QMBR, the expansion area has a major effect on performance. In addition, when QMBR and *Sq*MBR schemes were compared while increasing the quantization level, it was found to affect performance; the performance of *Sq*MBR was highest when the quantization level was 16 in Fig. 18. Accordingly, users' requests can be serviced faster by building indexes through compressing MBR keys using semi-quantization and reducing the space expanded on all four sides during quantization. The *Sq*MBR in Fig. 16 produced the same results with skewed real data as in the analysis assuming uniform data. For these reasons, *Sq*MBR method is superior to other compression schemes in performance using real data.

6. CONCLUSION AND FUTURE WORK

In LBS, a client asks a server for desired information and the server sends the result of the query. The server contains location information and searches for the location of the corresponding service. If the volume of location data is small, the service provision time decreases. Although the semi-quantization MBR scheme is an interesting and intuitive MBR compression scheme, it has received limited attention. In this paper, we introduced a new MBR compression scheme and proposed SQR-tree, a new spatial index that dramatically reduces MBR size. In particular, this study introduced the *Sq*MBR scheme, in which real MBR defines the enlarged space as FOR and reduces the corresponding region, and SQR-tree is the structure of the compressed index that adopts *Sq*MBR scheme. Traditional spatial index structures rarely use indexes constructed using MBR compression scheme. Existing compression schemes reduce the storage space required for keys in comparison with the original MBR. Nevertheless, our scheme decreases storage further and improves search performance by halving the enlargement region of QMBR.

The performance of our scheme was evaluated and compared with existing com-

pression schemes by implementing an algorithm in both SqMBR scheme and SQR-tree. In the experiment, the number of node accesses in SQR-tree, measured by changing the query region, the size of the nodes, and the quantization level, was distinct from those of the existing R-tree. The scheme also performed better than HMBR scheme, although the difference was small. The evaluation in terms of quantized levels compared our method with QMBR scheme using quantization. The methods differ sharply at level 16 due to the difference in the enlargement space in QMBR. The processing time for queries was equivalent to the number of node accesses, and QMBR scheme differed markedly from other schemes.

The results show that SQR-tree performed better than both R-tree and existing compression schemes, although the difference was small. In a structure in which the keys account for most of the index, compressing the keys reduces the height of the tree, as well as the size of the index, thereby reducing the search time. Therefore, we should consider the case in which we have to restore the compressed keys to their original values, a process that is also minimized using the algorithm proposed here. Our proposed scheme can be used to improve performance in areas that require a fast search, while having constraints on memory size or computation capability, such as in mobile devices. Accordingly, we plan to study performance improvement in those areas.

REFERENCES

1. J. Schiller and A. Voisard, *Location-based Services*, Morgan Kaufmann, San Francisco, 2004.
2. S. Y. Wu and K. T. Wu, "Dynamic data management for location based services in mobile environments," in *Proceedings of the 7th International Database Engineering and Applications Symposium*, 2003, pp. 180-191.
3. J. D. Kim, S. H. Moon, and J. O. Choi, "A spatial index using MBR compression and hashing technique for mobile map service," in *Proceedings of the 10th International Conference on Database Systems for Advanced Applications*, LNCS 3453, 2005, pp. 625-636.
4. K. H. Kim, S. K. Cha, and K. J. Kwon, "Optimizing multidimensional index trees for main memory access," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, Vol. 30, 2001, pp. 139-150.
5. Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima, "Spatial indexing of high-dimensional data based on relative approximation," *VLDB Journal*, Vol. 11, 2002, pp. 93-108.
6. J. Kim, S. J. Im, S. W. Kang, and C. S. Hwang, "Spatial index compression for location-based services based on a MBR semi-approximation scheme," in *Proceedings of the 7th International Conference on Advances in Web-Age Information Management*, LNCS 4016, 2006, pp. 26-35.
7. A. Guttman, "R-trees: a dynamic index structure for spatial searching," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, Vol. 14, 1984, pp. 47-57.
8. J. Goldstein, R. Ramakrishnan, and U. Shaft, "Compressing relations and indexes," in *Proceedings of the 14th IEEE Conference on Data Engineering*, 1998, pp. 370-379.

9. M. de Berg, M. van Kreveld, and M. Overmars, *Computational Geometry Algorithm and Applications*, 2nd ed., Springer, 2000, pp. 267-290.
10. "The R-tree portal, sequoia dataset," <http://www.rtreeportal.org/spatial.htm>.
11. H. Schwetman, "CSIM19: a powerful tool for building system models," in *Proceedings of the 33rd Conference on Winter Simulation*, 2001, pp. 250-255.



Jongwan Kim is a Ph.D. candidate at the Department of Computer Science and Engineering, Korea, Korea University. He received the M.S. degree in Computer Science from Soongsil University, Korea, 2001, and his B.S. degree in Business Administration from Shamyook University, Korea, 1991. He has more than 10 years field experiences as a developer and technician in object-oriented technology. He leads a project team, SEMO, for RFID smart edge monitoring system development and participates in data broadcasting project. His research interests include mobile and streaming data management, location-based services, sensor/RFID, and object-oriented technologies.



SeokJin Im received B.S. degree and the M.S. degree in Electronics from KookMin University in 1996 and 1998, South Korea. He was a researcher on Interconnection Modelling and Simulation at Hynix Semiconductor Research Center. He is currently a Ph.D. candidate in Computer Science and Engineering, Korea University, South Korea. His research interests include location-based services, moving objects database and data management for mobile computing.



Sang-Won Kang received his B.S. degree in Computer Science and his M.S. degree in Computer Science and Engineering from Korea University, Seoul, Korea in 1998 and 2003, respectively. He is currently working towards Ph.D. degree in Computer Science and Engineering from Korea University. Also, he is currently a researcher in the Research Institute of Computer Information Communication at Korea University. His academic interests include mobile computing systems, mobile data management, location dependent data, semantic prefetching & caching, data management in sensor network, indexing & query processing for moving objects, *etc.*



Chong-Sun Hwang received the M.S. degree in Mathematics from Korea University, Korea in 1970, and Ph.D. degree in Statics and Computer Science from University of Georgia I 1978. From 1978 to 1980, he was an Associate Professor at South Carolina Lander State University. He is currently a Full Professor in the Department of Computer Science and Engineering at Korea University, Seoul, Korea. Since 1995, he has been a dean in the Graduate School of Computer Science and Technology at Korea University. His research interests include distributed systems, distributed algorithms, and mobile computing systems.



SangKeun Lee received the B.S., M.S., and Ph.D. degrees in Computer Science and Engineering from Korea University, South Korea, in 1994, 1996, and 1999, respectively. He was a recipient of the Japan Society for the Promotion of Science (JSPS) Postdoctoral Fellowship in 2000. Since 2003, he has been an assistant/associate professor in the College of Information and Communication, Korea University, South Korea. His research interests include data management in mobile/pervasive computing systems, location-based information systems, XML databases, and data management in mobile ad hoc networks.