Performance Analysis of Object-Oriented Spatial Access Methods

Byungyeon Hwang

Department of Computer Science
Songsim University
Kyonggi, Puchon 422-743, Korea
byhwang@sicom.kaist.ac.kr

Songchun Moon

Department of Information and
Communications Engineering, KAIST
207-43, Cheongryangni, Dongdaemun
Seoul 130-012, Korea

Abstract

In geographic and CAD applications as well as in VLSI design, the practical need for access methods that allow for efficient spatial searching has increased considerably. In this paper, we show a performance comparison of the most promising spatial access methods, BR tree, R-tree, and R^+-tree, through performance results from analytical study. For search, BR tree produces the best performance with respect to the number of disk accesses since it generates the smallest number of nodes and avoids overlapping of rectangles. On the other hand, R^+-tree gives the smallest number of false drops because it avoids overlapping of rectangles unlike R-tree and contains more reduced empty space than BR tree. Furthermore, the experiment results are relatively similar to the results from analytical study.

1: Introduction

In geographic and CAD applications as well as in VLSI design, the practical need for access methods that allow for efficient spatial searching has increased considerably. Spatial data access methods provide a means of retrieving objects which are n dimensional points, lines, rectangles, or polygons. In particular, they optimize spatial queries which retrieve all points or rectangles which are enclosed in or overlap a specific search region.

Recently, there have been a substantial number of access methods proposed to manage point or solid data [1, 3, 7, 11, 13, 18]. However, performance studies of these access methods have often been analytical rather than based on real world implementations [6, 9, 15]. Also, the non-analytical performance studies consider the simulation results for a random data [2, 5, 8, 13].

In this paper, we will present a performance comparison of the most promising spatial access methods which are based on the approximation of a complex spatial object by the minimum bounding rectangle with the sides of the rectangle parallel to the axes of the data space. The most important property of this simple approximation is that a complex object is represented by a limited number of bytes. Although a lot of information is lost, minimum bounding rectangles of spatial objects preserve the most essential geometric properties of the object, i. e. the location of the object and the extension of the object in each axis. Our studies include the experiment results obtained from a real data as well as analytical study and simulation results.

The remainder of this paper is organized as follows. Section 2 describes the representative spatial access methods like R-tree [10], R^+-tree [19], and our spatial access method, BR tree [12]. Section 3 shows the performance results from analytical study for R-tree, R^+-tree, and BR tree. The comparison of performance from analytical study and experiment is shown in Section 4. Finally, conclusions appear in Section 5.

2: Spatial access methods

The access methods which we consider optimize queries that retrieve all the data objects overlapping a user-specified rectangular window **Q**. The definition of window query and allowable geometry is shown in *Fig. 1*. Here objects B, C, and D overlap the search region **Q**. In this study, we restrict boxes to 2-dimensional rectangles, specified 4 byte integer numbers, X_{min}, X_{max}, Y_{min}, and Y_{max}. In *Fig. 1*, for instance, boxes B through E are legal, whereas box A and F are not.

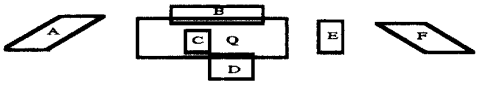


Fig. 1. Window query and allowable geometry

2.1: R-tree

R-tree is a height-balanced tree similar to the B-tree

0-8186-6430-4/94 $4.00

[4]. Let M be the maximum number of entries that fit in one node, and let $m \leq M/2$ be a parameter specifying the minimum number of entries in a node. The number of index record entries in each *leaf node* of an R-tree lies between m and M. The index entry has the form:

(I, tuple-identifier),

where *tuple-identifier* refers to a tuple in the database and I is an n-dimensional rectangle which is the bounding box of the spatial object indexed. I is represented as $I = (I_0, I_1, \ldots, I_{n-1})$. Here, n is the number of dimension and I_i is a closed bounded interval, [a, b], describing the extent of the object along dimension i. Each *non-leaf node* contains between m and M entries, each of which has the form:

(I, child-pointer),

where *child-pointer* is a pointer to a successor node in the next level of the R-tree, and I is the smallest rectangle that spatially contains the rectangle in the child node.

An example set of data rectangles and its corresponding R-tree are shown in *Fig. 2* and *Fig. 3*, respectively. In the Figures, for instance, the spatial objects D and H denote the 2-dimensional rectangles stored in a leaf node, and the region *R3* denotes the minimum rectangle that contains the objects in the descendant node.

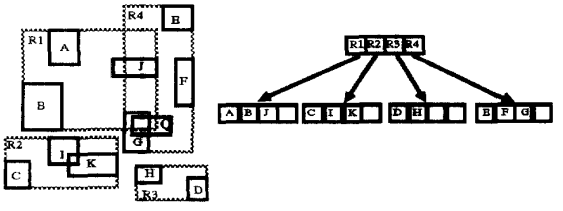


Fig. 2. Rectangles organized according to an R-tree

Fig. 3. An R-tree for the rectangles of *Fig. 2*

The term *branching factor*, called *fan-out*, can be used to specify the maximum number of entries that a node can have. For example, each node of an R-tree with branching factor 4 points to a maximum of 4 descendents (among non-leaf nodes) or 4 objects (among the leaves).

When we consider performance of R-tree with respect to the search, the concepts of *coverage* and *overlap* are important. The coverage of a level is defined as the total area of all the rectangles associated with the nodes of that level, and the overlap of a level is defined as the total area contained within two or more nodes. Obviously, efficient R-tree searching demands that both coverage and overlap be minimized, although overlap seems to be the more critical of the two issues. For a search window falling in the area of N overlapping leaves, in the worst case, N paths from the root to each of the overlapping leaves have to be followed slowing down the search from h to hN, where h is the height of the tree. The minimum coverage reduces the amount of empty space covered by the nodes, and the minimum overlap reduces the *false drop* problem which causes unnecessary searches of tree. In order to search **Q** in *Fig. 2*, both subtrees rooted at nodes R1 and R4 must be searched although only the latter will be true.

2.2: R⁺-tree

R⁺-tree may be thought as an extension of K-D-B tree [17] to cover spatial objects. Unlike R-tree, R⁺-tree does not limit the minimum number of entries contained in a leaf or non-leaf node. The index structure of R⁺-tree is the same as that of R-tree except that an object can be stored in several nodes because R⁺-tree avoids overlapping of rectangles. That is, in some cases, in which a given rectangle covering a spatial object at the leaf level overlaps with another rectangle, R⁺-tree decomposes it into a collection of non-overlapping sub-rectangles whose union makes up the original rectangle. Each *leaf node* of an R⁺-tree contains an entry of the form:

(Rectangle, tuple-identifier),

where *tuple-identifier* is an object identifier and is used to refer to an object in the database. *Rectangle* is used to describe the boundary of data objects. Each *non-leaf node* contains an entry of the form:

(Rectangle, child-pointer),

where *child-pointer* is a pointer to a lower level node of the tree and *Rectangle* is a representation of the enclosing rectangle.

The R⁺-tree and its rectangles corresponding to the rectangles in *Fig. 2* are shown in *Fig. 4* and *Fig. 5*.

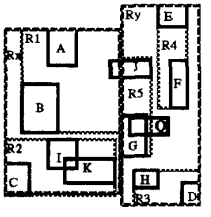


Fig. 4. Rectangles of *Fig. 2* on basis of an R⁺-tree

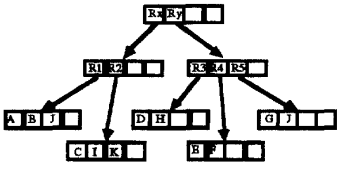


Fig. 5. An R⁺-tree for the rectangles of *Fig. 4*

In R⁺-tree all the pointers of a sub-rectangle clipped point the same object. R⁺-tree prevents overlapping of rectangles, but the depth of tree becomes high. For example, the height of the R-tree in *Fig. 3* is 2 and that of the R⁺-tree in *Fig. 5* is 3. Therefore, R⁺-tree requires large storage space.

2.3: BR tree

The BR tree is a new data structure for spatial indexing based on the *z-transform* [16]. In n-dimensional space, one point is represented by n coordinate values, i.e. n binary strings. For example, (6, 3, 7) represents a point in 3-dimensional space. Z-transform is the function that transforms an n-dimensional (n≥2) point into an one-dimensional point, called *z-value*. Z-value is obtained by extracting the first bit from each binary string and then enumerating these extracted bits, and then extracting the second bit, and so on. For example, if one point is n binary strings like ($b_{11}b_{12}...\ b_{1k}$, $b_{21}b_{22}...\ b_{2k}$, ..., $b_{n1}b_{n2}...\ b_{nk}$), where b_{ij} is the j^{th} bit of i^{th} dimension, z-value becomes $b_{11}b_{21}...\ b_{n1}b_{12}b_{22}...\ b_{n2}.....\ b_{1k}b_{2k}...\ b_{nk}$.

Z-transformed subspace, we call *z-area*, is represented by its z-value. The z-value for the z-area is constructed by interleaving the bits of each dimension, which describes the range of values covered by the z-area, bit by bit. Assume that the horizontal and vertical axes of a 2-dimensional space are X and Y, respectively. Suppose that the ranges of X and Y values covered by the z-area are described by ($4 \le X \le 5$, $0 \le Y \le 3$). In binary, these ranges are ($\underline{100} \le X \le \underline{101}$, $\underline{000} \le Y \le \underline{011}$). The z-area is expressed by the common prefix of each range: (10, 0). When the prefix value of X begins first, interleaving these bits yields a z-value of 100.

In a BR tree, each z-area corresponds to one disk block. When we represent a set of z-areas by one-dimensional array named r, each 1-bit entry of array r should be changed to the address of disk block in order to index disk blocks. If the address of disk block is 0, i.e. null, this means that the partition does not exist. When the index itself is stored in disk and the size of one disk block is fixed, in order to insert many objects of real space, more than one disk block can be required. If these blocks are stored in continuous blocks, search operation becomes poor by sequential search. Therefore, in order to retrieve indices stored in more than one disk block effectively, an index structure of tree form is required.

In a BR tree, data objects are stored at leaf nodes, and the subregions in the children nodes are stored at non-leaf nodes. An example set of data rectangles and its corresponding BR tree are shown in *Fig. 6* and *Fig. 7*, respectively. The branching factor is assumed to be 4. In the Figures, for instance, the spatial objects A and B are the 2-dimensional rectangles intersected or included in the z-area whose z-value is 01. The second entry in the non-leaf node indicates the block address of the leaf node containing the objects A and B.

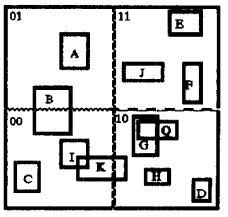


Fig. 6. Rectangles organized according to a BR tree

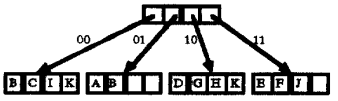


Fig. 7. A BR tree for the rectangles of *Fig. 6*

Each *leaf node* in a BR tree contains index record entries of the form:

$$(Oid, R),$$

where *Oid* is an object identifier which is used to refer to an object and *R* represents the boundary of the objects. For the object B in *Fig. 7*, *Oid* is the pointer to the disk block containing the object, and *R* is the values of coordinates enclosing the object. Each *non-leaf node* contains entries of the form:

$$(cp_1, cp_2, ..., cp_n),$$

where each cp_i is a pointer to a child node. Let *M* be the maximum number of entries in a non-leaf node. The BR tree satisfies the following properties.

1. The *root node* has between *2* and *M* children.
2. Every *non-leaf node* has between *1* and *M* children unless it is the root.
3. Every *leaf node* contains between *0* and $\lfloor M/((p+2nd)/p) \rfloor$ index records, where *p* is the size of pointer, *n* is the number of dimension, and *d* is the size of each element in the values of coordinates.
4. *Non-leaf node* is not necessarily the smallest rectangle that spatially contains the rectangles in the child node because the pointer values in each node represent the location of a region. For the first entry of non-leaf node in *Fig. 7*, the first 0 and the second 0 denote the left and low side of partitioned space, respectively. The null pointer in *non-leaf node* means that a partition does not exist.

3: Analytical study for point queries

In this section, we analyze the performance of BR trees, R^+-trees, and R-trees with respect to the effects of the number of disk accesses for point queries. For these access methods, the search performance depends on not only the data itself, i.e. the number of objects, sizes, and locations, but also the sequence of objects inserted in the access methods [6]. To avoid the effects of the insertion

sequences and split routines on the number of disk accesses and storage overhead, we perform a best-case comparison: Given all the data objects in advance, the non-leaf nodes of each access method have minimal *coverage* and *overlap* in the split algorithm. The coverage of a level is defined to be the total area of all the rectangles associated with the nodes of that level, and the overlap of a level is defined to be the total area contained within two or more non-leaf nodes of the tree index structure.

The largest portion of the analysis is devoted in calculating the characteristics of the optimal fathers having minimal coverage and overlap in the split algorithm. For each method we consider the cases where all the nodes are full. To simplify the analysis, we assume that the data objects belong to a few classes and each class consists of objects of the same size since the analytical method consisting objects of the different size is impossible. This is true in the engineering application like VLSI designs, where there are a few types of elements, e.g. busses, NOR-gates etc., with many instances of the elements in the design. Geographic applications follow this assumption with small deviations. For example, the states of the U. S. A. have similar sizes, with few exceptions. Furthermore, we assume that the objects are uniformly distributed. A characteristic of the uniform distribution for the objects is that the optimal father nodes will also be uniformly distributed.

3.1: Method for analytical study

The proofs of the forthcoming analyses can be derived more easily if we consider rectangular objects as points in a 4-dimensional space [11]. For a rectangular object aligned with the axes, four coordinates, i.e. the x and y coordinates of the lower-left and upper-right corners, are enough to uniquely determine it. Since 4-dimensional spaces are impossible to be illustrated, in this paper we consider line segments (1-d objects) instead of rectangles (2-d objects), and we transform the segments into points in a 2-d space. Each segment is uniquely determined by (*Xstart*, *Xend*), the coordinates of its start and end points. Obtaining formulas and results for line segments is a first step to the analysis of 2-d rectangular objects, or even rectangles of higher dimensionality.

In the case of line segments, the "screen" collapses to a line segment, which starts at 0 and ends at 1. An example set of line segments and their 2-d representation are shown in *Fig. 8* and *Fig. 9*, respectively. In the Figures, some important observations with respect to the transformed space are as follows.

(1) There are no points below the diagonal in *Fig. 9* since *Xstart* ≤ *Xend*.
(2) In *Fig. 9*, line segments of equal size such as B and C are represented by points that lie on a line parallel to the diagonal. Zero-size segment, i.e. point, like F, is represented by point on the diagonal.
(3) Line segments not entirely within the screen like A in *Fig. 8* are allowed. In the analysis we consider their remainder after clipping (solid part of A in *Fig. 8*).
(4) Points outside the shaded area in *Fig. 9* are of no interest because the corresponding segments do not intersect with the screen.

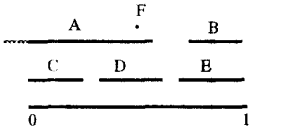


Fig. 8. Some line segments

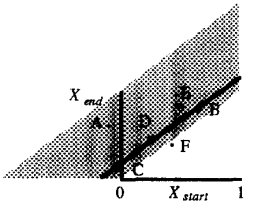


Fig. 9. 2-d representation of the line segments of *Fig. 8*

(5) The segments are *semi-open*. That is, they contain their leftmost end, but not the rightmost one. The screen is *semi-open*, too, including "0", but not "1". Query segments are closed.
(6) The segments covering a given point X_0 of the screen are transformed to points in a shaded area as shown in *Fig. 10*. The shaded area in *Fig. 11* corresponds to all the segments intersecting with the segment $S(X_1, X_2)$.

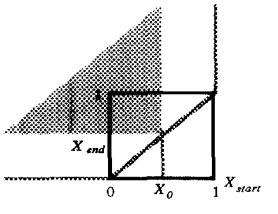


Fig. 10. Segments covering the point X_0

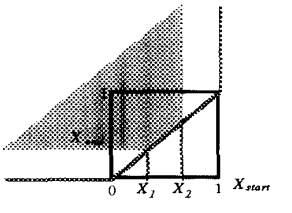
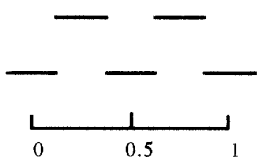


Fig. 11. Segments intersecting with the segment $S(X_1, X_2)$

To make the analysis tractable, we assume that the line segments of a given size need not be totally within the screen. These segments divide the interval $(-\sigma, 1)$ to $N+1$ equal subintervals, where N is the number of segments and σ is their size. An example set of 5 segments of size 0.25, which are uniformly distributed, and their corresponding 2-d transformations are shown in *Fig. 12*

and *Fig. 13*, respectively. In *Fig. 13*, 5 segments divide the interval (-0.25, 1) to 6 equal subintervals.



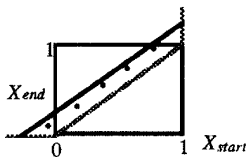


Fig. 12. Segments of size 0.25, on the screen

Fig. 13. 2-d transformations of the segments of *Fig. 12*

Definition: For a given point, let overlap O_v be the number of segments that contain it.

Table 1 indicates the parameters common to all the forthcoming analyses.

Table 1 Parameters of the analysis

Parameter	Description
C	the capacity of the data pages (=data objects per page)
F_{br}	the fanout of the internal nodes of the BR tree
F_{r+}	the fanout of the internal nodes of the R⁺-tree
F_r	the fanout of the internal nodes of the R-tree
H_{br}	the height of a BR tree
H_{r+}	the height of an R⁺-tree
H_r	the height of an R-tree

3.2: Analytical study for BR tree

We consider N segments of size σ, which are uniformly distributed as described before. An important lemma is as follows.

Lemma: Given N segments of size σ, which are uniformly distributed on the screen, a line segment (*query segment*) of size q intersects with $intsect(N, \sigma, q)$ segments, where

$$intsect(N, \sigma, q) = \frac{(\sigma + q)}{(1 + \sigma)} (N+1) \quad 0 \le q \le 1 \quad (1)$$

Proof: Consider *Fig. 14*. Projecting the line AB on the horizontal axis, we have the line AB', of size $1+\sigma$. The query region (shaded area) intersects the line AB on a segment CD, whose projection $C'D'$ is of a size $\sigma+q$. The fraction of the intersected segments, i.e. circles on line CD or circles on line $C'D'$, is $length(CD) / length(AB) = (\sigma+q) / (1+\sigma)$. Since the line AB is divided into $N+1$ equal subintervals by the circles, the line CD will contain on the average

$$\frac{(\sigma + q)}{(1 + \sigma)} (N+1)$$

circles, which is exactly the number of segments that the query segment q will intersect.

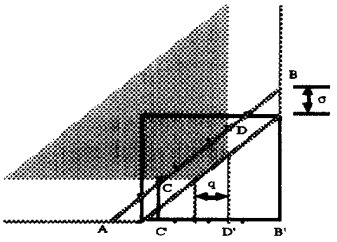


Fig. 14. N segments of size σ represented as circles

Notice that the formula does not depend on the position of the query segment on the screen since we have defined the uniform distribution of the N segments. For q=0, that is, the query segment is a point, we have the formula for the overlap.

Corollary: Given N segments with size σ, which are uniformly distributed on the screen, the overlap is constant, and given by the formula:

$$O_v(N, \sigma) = \frac{\sigma}{(1 + \sigma)} (N+1) \quad (2)$$

From (1) and (2) we have $intsect(N, \sigma, q) = O_v(N, \sigma) + q(N+1-O_v(N, \sigma))$.

For example, consider the case of N=5 and σ=0.5. For the segments shown in *Fig. 15*, the overlap is [0.5 / (1+0.5)] × (5+1) = 2. Every point on the screen is covered by 2 segments.

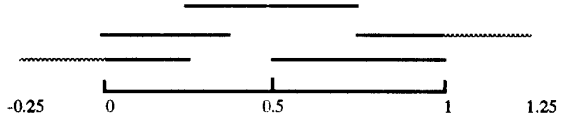


Fig. 15. 5 segments of size 0.5 each

Assume that we have N segments of size σ, which are uniformly distributed on the screen, as explained before. Let H_{br} be the height of a BR tree, which is assumed to be full, that is, every data page contains C entries, each internal node has F_{br} sons. We assume that level 1 is the first level above the data objects and the root is at level $H_{br}+1$.

The total number of data pages will be F^{Hbr} dividing

the screen into F^{Hbr} intervals. Due to the uniformity assumption, each interval will be of a size $1/F^{Hbr}$. Each page corresponds to one such interval. The segments that intersect such an interval will be entered in the corresponding page. Since each page is full, we have

$$C = intsect(N, \sigma, 1/F^{H_{br}}) \text{ or } C = O + \frac{1}{F^{H_{br}}}(N+1-O)$$

$$H_{br} = \log_{F_{br}} \frac{N+1-O}{C-O}$$

where O is the overlap $O = O_v(N, \sigma)$ of the given set of segments.

The total number of disk accesses *br_da* for a point query is the height incremented by one, to account for the retrieval of the data page. The root of the tree is assumed to be on the disk. Thus, the *br_da* for a point query is as follows.

$$br\_da = 1 + \log_{F_{br}} \frac{N+1-O}{C-O} \quad (3a)$$

or, since $N >> 1$ and $N >> O$,

$$\boxed{br\_da \approx 1 + \log_{F_{br}} \frac{N}{C-O}} \quad (3b)$$

3.3: Analytical study for R-tree

In the same way as above we do the analytical study for R-tree. The tree is assumed to be full. Thus, the segments are grouped in N/C pages, in groups of size C. Each page is characterized by the "minimum enclosing segment", which covers all the segments in this page. This segment corresponds to a point in the transformed space. For example, in *Fig. 16*, "A" is the minimum enclosing segment for the segments A_1 and A_2.

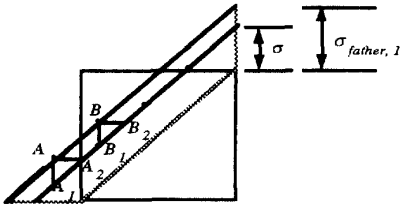


Fig. 16. Illustration of fathers of level 1 (C=2)

These segments will be referred as by "fathers of level 1". It can be proven in [6] and $\sigma_{father,\,1}$ of equal size, which are uniformly distributed, is as follows.

$$\sigma_{father,\,1} = \frac{(1+\sigma)}{(N+1)}(C-1) + \sigma \quad (4)$$

Data pages are grouped in $N/(CF_r)$ groups, each group containing F_r data pages (and therefore CF_r segments), to form the lowest level of internal nodes of the R-tree ("fathers of level 2"). In general, "fathers of level i" have size

$$\sigma_{father,\,i} = \frac{(1+\sigma)}{(N+1)}(CF_r^{\,i-1} - 1) + \sigma \quad (5)$$

and are uniformly distributed, in the sense described before. The root of the R-tree corresponds to a father of level H_r+1.

The overlap $O_{v,\,father,\,i}$ for fathers of level i is the number of fathers of level i containing a point on the screen. Since the number $N_{father,\,i}$ of i-level fathers is

$$N_{father,\,i} = \frac{N}{CF_r^{\,i-1}} \quad (6)$$

their overlap $O_{v,\,father,\,i}$ is, from the corollary (Eq. (2))

$$O_{v,\,father,\,i} = \frac{\sigma_{father,\,i}}{1+\sigma_{father,\,i}}(N_{father,\,i}+1) \quad (7)$$

which becomes

$$O_{v,\,father,\,i} = 1 + \frac{O_v - 1}{CF_r^{\,i-1}} \quad (8)$$

To find the number of disk accesses *r_da* in answering a point query, we have to add all the fathers of any level, which cover the query point. That is

$$r\_da = \sum_{i=1}^{H_r+1} O_{v,\,father,\,i} \quad (9)$$

where H_r is the height of the full R-tree.

$$H_r = \log_{F_r} \frac{N}{C} \quad (10)$$

Then, (8) and (10) give when H_r is an integer

$$\boxed{r\_da = H_r + 1 + \frac{O_v - 1}{C}\,\frac{F_r}{F_r - 1}\left(1 - \frac{1}{F^{H_r+1}}\right)} \quad (11)$$

3.4: Analytical study for R$^+$-tree

In the same way as above we do the analytical study for R$^+$-tree. The R$^+$-tree is similar to the BR tree since they prevent overlapping of rectangles in the non-leaf nodes of a tree by clipping objects which cause overlapping region. However, unlike BR trees, R$^+$-trees need the values of coordinates in representing a region of non-leaf nodes. Therefore, the fan-out of an R$^+$-tree is smaller than that of a BR tree. That is, R$^+$-trees have the smaller entries than BR trees in each non-leaf node. Using the same arguments as BR trees, we have:

$$r^+\_da \approx 1 + \log_{F_{r+}} \frac{N}{C\text{-}O} \qquad (12)$$

3.5: Analytical results for point queries

Based on the formulas described in the previous section, we obtain some results that indicate how BR trees are compared to R⁺-trees and R-trees. The following values are assumed.

size_page = 1,024 bytes, *size_rectangle* = 16 bytes
size_pointer = 4 bytes, *size_header* = 24 bytes

Given these numbers we can derive the fan-out F_{r+}, F_r, and the capacity C of leaf pages. We assume that the first 24 bytes of a page are taken for the header.

$$F_{r+} = F_r = C = \left\lfloor \frac{(size\_page - size\_header)}{(size\_rectangle + size\_pointer)} \right\rfloor = \left\lfloor \frac{(1{,}024\text{-}24)}{(16+4)} \right\rfloor = 50$$

However, unlike R-tree and R⁺-tree, BR tree does not need the values of coordinates in representing a region of a non-leaf node since the region is simply represented by a relative pointer value. Therefore, F_{br} is as follows.

$$F_{br} = \left\lfloor \frac{(size\_page - size\_header)}{size\_pointer} \right\rfloor = \left\lfloor \frac{(1{,}024\text{-}24)}{4} \right\rfloor = 250$$

Fig. 17 gives the disk accesses as a function of N segments in the case where overlap O = 40. *Fig. 18* gives the same values as a function of O in the case where N = 100,000.

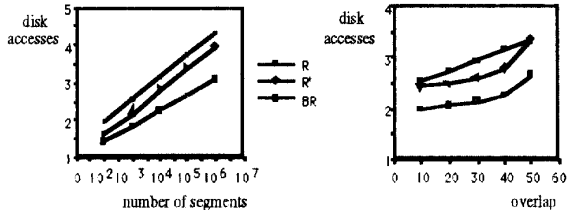


Fig. 17. Disk access for N

Fig. 18. Disk access for O

As expected the R-tree always performs worse than the BR tree and R⁺-tree since the excessive searching by overlapping of non-leaf nodes is required. Also, the BR tree has always better performance than the R⁺-tree since it contains more entries than the R⁺-tree in non-leaf nodes.

In three access methods the number of disk accesses increases with the number of segments since the height of the tree increases. Finally, increasing overlap also increases the number of pages searched; in R-trees this is due to multiple paths that have to be followed till the leaves have been reached, while in the BR tree and R⁺-tree it is due to increasing number of sub-segments that have to be created because of splits.

4: Comparison of performance from analytical study and experiment

In this section, we compare performance results from analytical study and those from experiment [12] with respect to the number of disk accesses for point queries in order to verify the correctness of the analytical model. The experiment results are relatively similar to the results from analytical study. For example, as expected in the results from analytical study, BR tree produces the best search performance in simulation results since it induces the smallest number of nodes and avoids overlapping of rectangles of non-leaf nodes. Furthermore, R-tree always performs worse than the BR tree and R⁺-tree since the excessive searching by overlapping of non-leaf nodes is required. In three access methods, the number of disk accesses increases with the number of objects since the height of the tree increases. Unlike R⁺-tree, BR tree does not need downward propagation for split in the non-leaf nodes. Therefore, the depth of BR tree is smaller than that of R⁺-tree and the experiment search performance of BR tree is better than the analytical search performance.

We consider a quantitative comparison for the analytical study and simulation. *Table 2* gives a list of performance results from analytical study and those from simulation for BR tree and in addition a set of relative errors between them. Here the error rate is computed as follows.

Error Rate = [Max(E, A) - Min(E, A)] / Max(E, A)
where E and A indicate the performance results from simulation and those from analytical study, respectively.

Table 2 Comparison of performance from analytical study and experiment

num_object / num_disk	513	1058	2011	4994
Theory	1.71	1.84	1.94	2.14
Experiment	1.51	1.66	2.0	2.2
Error Rate	11.7 percent	9.8 percent	3.0 percent	2.7 percent

From the table, we can see that the error rates between the performance results from analytical study and those from simulation with respect to the number of disk accesses are about 2-11 percent. Unlike in simulation results, we assumed that all the leaf nodes of each access method are full and the non-leaf nodes have minimal

coverage and minimal overlap in analytical study. Therefore, the quantitative performance values from analytical study may be different from those from simulation. The relative performance comparison is more important than the quantitative performance comparison. In both the analytical study and simulation, BR tree always produces the best search performance and R-tree always performs worst. As a result, it is shown that the performance results from simulation and those from analytical study relatively agree well.

5: Conclusions

In this paper, our spatial access method, called BR tree, is compared with the two representative spatial access methods, R-tree and R^+-tree, through the performance results from analytical study and those from simulation approach for a VLSI data.

For point queries, BR tree has always better performance than R-tree and R^+-tree since it contains more entries than R-tree and R^+-tree in non-leaf nodes. In three access methods, the number of disk accesses increases with the number of objects since the height of the tree increases. Furthermore, increasing overlap also increases the number of pages searched; in R-tree this is due to multiple paths that have to be followed till the leaves have been reached, while in BR tree and R^+-tree it is due to increasing number of nodes that have to be created because of splits.

For region queries, in general, BR tree gives the smallest number of disk accesses since it avoids overlapping of rectangles in non-leaf nodes and decreases the number of nodes. However, when the size of a query window is above 10 percent of each dimension, the empty space of BR tree increases, and R-tree gives the smallest number of disk accesses.

On the other hand, R^+-tree gives the smallest number of false drops because it avoids overlapping of rectangles unlike R-tree and contains more reduced empty space than BR tree.

In the analytical study for spatial access methods, we considered line segments (1-dimensional objects) instead of boxes (2-dimensional objects), and transformed the segments into points in a 2-dimensional space. It would be desirable to extend the analytical study to the case of arbitrary dimensions. This will allow us to examine objects in 2-dimensional spaces which are found in many applications.

References

[1] J. Banerjee and W. Kim, "Supporting VLSI Geometry Operations in a Database System," Proc. of 2nd Int. Conf. on Data Engineering, 1986, 409-415.

[2] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles," Proc. of ACM SIGMOD, 1990, 322-331.

[3] T. Brinkhoff, H. P. Kriegel, and B. Seeger, "Efficient Processing of Spatial Joins Using R-trees," Proc. of ACM SIGMOD, 1993, 237-246.

[4] D. Comer, "The Ubiquitous B-Tree," ACM Computing Surveys, Vol. 11, No. 2, Jun. 1979, 121-137.

[5] C. Faloutsos and Y. Rong, "DOT: A Spatial Access Method Using Fractals," Proc. of 7th Int. Conf. on Data Engineering, 1991, 152-159.

[6] C. Faloutsos, T. Sellis, and N. Roussopoulos, "Analysis of Object Oriented Spatial Access Methods," Proc. of ACM SIGMOD, 1987, 426-439.

[7] M. Freeston, "The BANG file: A New Kind of Grid File," Proc. of ACM SIGMOD, 1987, 260-269.

[8] D. Greene, "An Implementation and Performance Analysis of Spatial Data Access," Proc. of 5th Int. Conf. on Data Engineering, 1989, 606-615.

[9] O. Gunther and H. Noltemeier, "Spatial Database Indices for Large Extended Objects," Proc. of 7th Int. Conf. on Data Engineering, 1991, 520-526.

[10] A. Guttman, "R-trees: A Dynamic Index Structure for Spatial Searching," Proc. of ACM SIGMOD, 1984, 47-57.

[11] K. Hinrichs and J. Nievergelt, "The Grid File: A Data Structure to Support Proximity Queries on Spatial Objects," Technical Report 54, ETH, Zurich, 1983.

[12] B. Y. Hwang, B. W. Kim, and S. C. Moon, "Efficient Access Method for Multi-Dimensional Complex Objects in Spatial Databases: BR Tree," Journal of Microprocessing and Microprogramming, Vol. 32, No.1-5, 1991, 765-772.

[13] W. Lu and J. Han, "Distance-Associated Join Indices for Spatial Range Search," Proc. of 8th Int. Conf. on Data Engineering, 1992, 284-292.

[14] W. Mendenhall, Introduction to Probability and Statistics, Duxbury Press, 1987.

[15] J. Nievergelt, H. Hinterberger, and K. C. Sevcik, "The Grid File: An Adaptable, Symmetric Multikey File Structure," ACM Trans. on Database Systems, Vol. 9, No. 1, 1984, 38-71.

[16] J. A. Orenstein, "Spatial Query Processing in An Object-oriented Database System," Proc. of ACM SIGMOD, 1986, 326-336.

[17] J. T. Robinson, "The K-D-B Tree: A Search Structure for Large Multidimensional Dynamic Indexes," Proc. of ACM SIGMOD, 1981, 10-18.

[18] B. Seeger and H. P. Kriegel, "The Buddy Tree: An Efficient and Robust Access Method for Spatial Databases," Proc. of 16th Int. Conf. on Very Large Databases, 1990, 590-601.

[19] T. Sellis, N. Roussopoulos, and C. Faloutsos, "The R^+-tree: A Dynamic Index for Multi-dimensional Objects," Proc. of 13th Int. Conf. on Very Large Databases, 1987, 507-518.