Parametric R-Tree: An Index Structure for Moving Objects*

Mengchu Cai
IBM Silicon Valley Laboratory
555 Bailey Ave.
San Jose, CA 95141, USA
mcai@us.ibm.com

Peter Revesz
Dept. of Computer Science and Engineering
University of Nebraska–Lincoln
Lincoln, NE 68588, USA
revesz@cse.unl.edu

ABSTRACT

We describe an indexing method for *parametric rectangles* that were recently proposed in [6] to represent moving objects. Parametric rectangles are a more natural representation of moving objects than *moving points*. Our indexing method extends R-trees, with the following important modifications among others: (i) definition of parametric rectangle trees, or PR-trees (ii) searching a *PR*-tree for intersection queries (iii) insertion into *PR*-trees (iv) deletion from *PR*-trees. These modified operations need new algorithms for finding a minimum bounding parametric rectangle (MBPR) of a set of parametric rectangles and a new insertion and splitting criteria and algorithms. Experiments show that *PR*-trees provide a significant improvement over R-trees for intersection queries with moving rectangles.

1. INTRODUCTION

Applications such as weather forecasting and air traffic control require the monitoring of the continuous change in the shape and the position of a large number of moving objects. Such novel applications pose many challenges because relational databases cannot model continuous change, and spatial indexing techniques like R-trees [11], R*-trees [2] and R+-trees [18] cannot efficiently index moving objects.

Some recent work propose to represent moving objects by moving points, for example [7, 9, 19]. Other work also consider the indexing of moving points [1, 15, 16, 20]. However, a limitation of moving points is that they do not represent the spatial extents of objects and therefore cannot express some natural queries that involves intersection of the spatial extents. For example, if each airplane and cloud is represented as a moving point, then a simple query like "*Find all the airplanes that are currently traveling within some cloud*" cannot be expressed and evaluated.

Constraint database [13] systems like MLPQ/GIS [14], DEDALE [10] and CCUBE [5] with linear constraints of the spatial and temporal dimensions can represent moving objects with their changing extents, but their data representation does not allow efficient indexing except in special cases [4, 12].

Our work is based on the PReSTO [6] database system, which uses sets of *parametric rectangles* for modeling moving objects. This also allows representing moving objects with their changing extents. For example, parametric rectangles could model a set of growing or shrinking clouds that cannot be modeled by moving points. The PReSTO database system can also express relational algebra queries (like the query above).

In this paper we describe a new indexing method called *PR-tree* for parametric rectangles. While the overall method is based on the main ideas of R-trees, several important considerations are special to *PR*-trees, including new definitions/algorithms for finding the parametric bounding box for a set of other parametric bounding boxes, choosing a subtree in a *PR*-tree, and a splitting criterion.

The paper is structured as follows. Section 2 gives the definition and background of parametric rectangles. Section 3 describes the *PR*-tree index structure and an efficient algorithm to compute the minimum bounding parametric rectangle of a set of moving objects. Section 4 describes the *PR*-tree search algorithm. Section 5 presents the insertion and deletion algorithms of the *PR*-tree. Section 6 compares the performance of *PR*-trees with *R*-trees. Finally, Section 7 covers related work.

2. BACKGROUND: PARAMETRIC RECTANGLES

We consider a database of moving objects that change their position and shape continuously over time. Assume that the moving objects are modeled as a set of parametric rectangles [6]. A parametric rectangle (R) is a multidimensional moving rectangle defined as follows:

$$R = (I_1, \ldots, I_d,\ t^{\lfloor},\ t^{\rfloor})$$

where d is the number of dimensions, I_i is a closed bounded interval in the ith dimension such that the lower and the upper bound of I_i denoted by $x_i^{\lfloor}$ and $x_i^{\rfloor}$ are linear functions of time t where $t \in [t^{\lfloor},\ t^{\rfloor}]$ and the start time $t^{\lfloor}$ and the end time $t^{\rfloor}$ are two rational numbers. The projection of R

*The second author was supported by NSF grant IRI-9625055 and a Gallup Research Professorship. Contacts: revesz@cse.unl.edu

ADVANCES IN DATA MANAGEMENT 2000
Krithi Ramamritham, T.M. Vijayaraman (Editors)
Tata McGraw-Hill Publishing Company Ltd.

on the x_i dimension, denoted by Π_{x_i}, is a one dimensional parametric rectangle or interval $(I_i,\ t^{\lfloor}, t^{\rfloor})$.

The semantics of a d-dimensional parametric rectangle r is a polyhedron P in the $d+1$ space. Given two parametric rectangles $r1$ and $r2$, we say that $r1$ includes $r2$ if the corresponding polyhedron of $r1$ includes that of $r2$.

3. THE PARAMETRIC R-TREE INDEX STRUCTURE

Our goal is to develop an index structure to answer efficiently queries of the form:
"Report all objects that intersect a moving rectangle R". R is also represented by a parametric rectangle. The index structure allows the database to be dynamic, that is, to insert a new object or to delete an old one from the database.

The parametric R-tree (PR-tree) we present is an R-tree [11] like structure for multidimensional moving objects. Like an R-tree, a PR-tree is a height-balanced tree and each node has between $M/2$ and M children, where M is a constant depended on the page size. However, while each node of an R-tree is associated with a minimum bounding rectangle that represents the stationary position and extents of a spatial object, in contrast, each node of a PR-tree is associated with a *minimum bounding parametric rectangle* which is defined as follows:

DEFINITION 3.1. *Suppose S is a set of d-dimensional parametric rectangles, then we call a parametric rectangle r the* minimum bounding parametric rectangle *of S if and only if*

1. *r contains all parametric rectangles in S.*
2. *The area of the project of r onto the (x_i, t) space is minimized for each $i = 1, \ldots, d$.*

EXAMPLE 3.1. *Table 1 represents eight moving objects (r4 – r11) and their bounding boxes r1, r2 and r3 by parametric rectangles. Figure 1 shows the snapshots of the moving objects (shaded) and their bounding boxes at $t = 0$ and $t = 10$. $r6$ and $r8$ are not shown in the snapshot at $t = 0$ since they do not exist at $t = 0$. For the same reason, $r7$ is not shown in the snapshot of $t = 10$. Figure 2 shows the eight moving rectangles organized in a* PR*-tree with $M = 3$.*

	$x^{\lfloor}$	$x^{\rfloor}$	$y^{\lfloor}$	$y^{\rfloor}$	$t^{\lfloor}$	$t^{\rfloor}$
r1	$30+7t$	$90+7t$	$50+5t$	$100+6t$	0	10
r2	0	50	$10+5t$	$55+5t$	0	10
r3	$60-6t$	$100-5t$	0	$40+t$	0	10
r4	$30+7t$	$50+7t$	$80+6t$	$100+6t$	0	10
r5	$30+12t$	$40+12t$	$50+5t$	$65+5t$	0	10
r6	$75+6t$	$90+7t$	$70+8t$	$80+8t$	2	10
r7	0	15	$40+5t$	$55+5t$	0	9
r8	0	12	$20+4t$	$40+4t$	1	10
r9	30	50	$10+7t$	$20+7t$	0	10
r10	$80-5t$	$100-5t$	$2t$	$20+3t$	0	10
r11	$60-6t$	$70-6t$	$30-3t$	$40-2t$	0	10

Table 1: Moving objects and their bouding boxes represented by parametric rectangles

Now let us consider how to compute the minimum bounding parametric rectangle. Suppose that S is a set of d-dimensional parametric rectangles. Let R be the MBPR

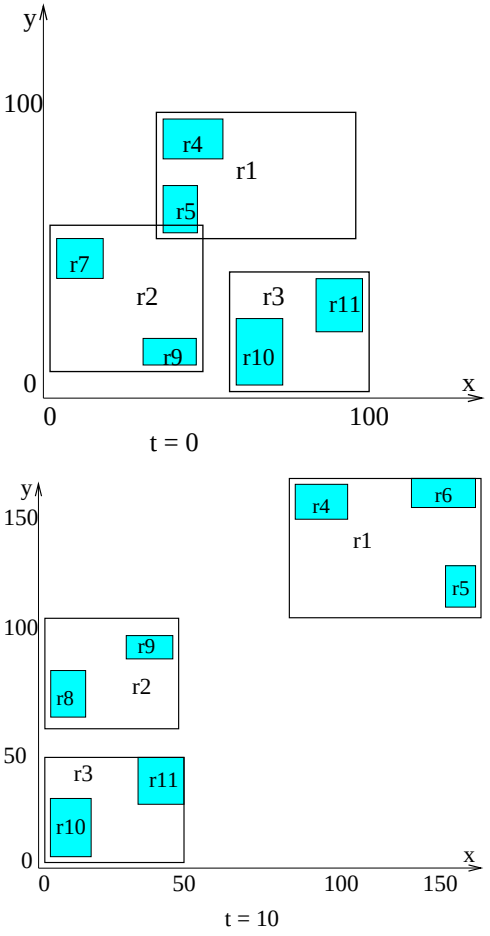

Figure 1: Two snapshots of the moving objects and their bounding boxes

of S. It is easy to see that the start time of R is the minimum of the start times of the parametric rectangles in S and the end time of R is the maximum of the end times of the parametric rectangles in S. Let t_{min} and t_{max} denoted the start and the end time of R, then

$$t_{min} = \min_{r \in S}(r.t^{\lfloor}),\quad t_{max} = \max_{r \in S}(r.t^{\rfloor})$$

So the main task of computing R is to compute the functions for the lower and the upper bounds of R in x_i dimension, for each $i = 1, \ldots, d$. Since the projection of each parametric rectangle in S onto the $(x_i,\ t)$ space corresponds to a trapezium with four extreme points as shown in Figure 3 (left), the projection of S onto the (x_i, t) space corresponds a set S_i of $4|S|$ extreme points in the $(x_i,\ t)$ space. Let H_i be the convex hull of S_i. We will show that the lower and the upper bounds of R for the x_i dimension are extensions of some edges of the convex hull H_i, which can be computed efficiently [8]. For example, the polygon in thin solid line in Figure 3 is the convex hull of four parametric rectangles in the (x_i, t) space, and the trapezium in bold line in Figure 3 (right) is their MBPR projected on the (x_i, t) space.

LEMMA 3.1. *For each x_i, let S_i be the projection of a set*

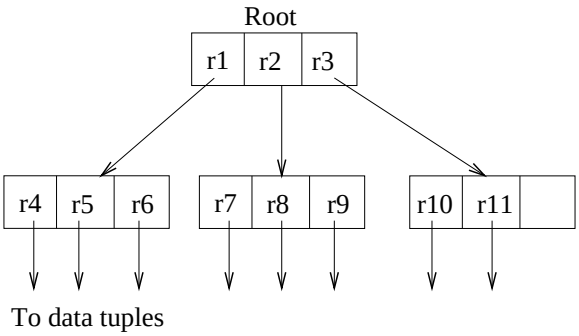

Figure 2: A *PR*-tree

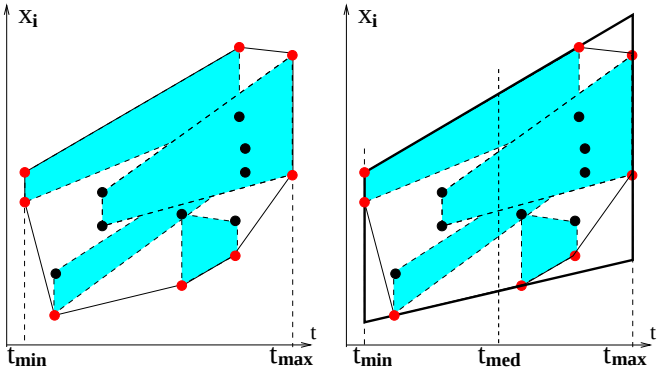

Figure 3: The convex hull and MBPR of four parametric rectangles

S of parametric rectangles onto the (x_i, t) space, then the lower and the upper bounds of the MBPR of S projected onto (x_i, t) overlaps some edge of the convex hull of S_i.

LEMMA 3.2. *The lower and the upper bounds of the minimum bounding parametric rectangle of S projected onto (x_i, t) are the extensions of the edges of H_i that intersect the median line $t = t_{med}$, where*

$$t_{med} = \frac{t_{min} + t_{max}}{2}$$

.

Figure 3 shows in the x_i dimension the bounds of the MBPR of a set of parametric rectangles are extension of the edges of the convex hull that intersect the line $t = t_{med}$.

THEOREM 3.1. *The minimum bounding parametric rectangle of M number of d-dimensional parametric rectangles can be computed in $O(dM \log M)$ time.*

Proof: Let S be a set of M number of d-dimensional parametric rectangles. It is easy to see t_{max} and t_{min} can be computed in $O(M)$ time. For each $i = 1, \ldots, d$, let S_i be the projection of S onto the $(x_i,\ t)$ space, we can compute the corresponding extreme points of S_i in $O(M)$ by subtituting the start and the end times of each parametric rectangle in S to the functions. The convex hull of S_i can be computed in $O(M \log M)$ time [8]. We can also find the edges of H_i that intersect with t_{med} in $O(M)$ time. By Lemmas 3.1 and

Input: S (a set of parametric rectangles)
Output: R_S, the MBPR of S
1. $t_{min} = \min_{r \in S}(r.t^{\lfloor})$
2. $t_{max} = \max_{r \in S}(r.t^{\rfloor})$
3. $t_{med} = (t_{min} + t_{max})/2$
4. **FOR** each dimension x_i **DO**
5. Find S_i the set of extreme points of the $\Pi_{(x_i,t)}S$.
6. Compute the convex hull H_i of S_i
7. Find the edges of H_i that intersect with t_{med}
8. Construct $x_i^{\lfloor}$ and $x_i^{\rfloor}$
9. **END-FOR**

Figure 4: Algorithm for Computing MBPR

3.2, the functions of the lower and the upper bounds of the MBPR R of S in x_i are the functions of the edges of H_i that intersect with t_{med}. Therefore the bounds of R in x_i can be computed in $O(M) + O(M \log M) + O(M) = O(M \log M)$ time.

For d dimensions, the MBPR of S can be computed in $O(dM \log M)$ time. The algorithm is outlined in Figure 4. □

4. SEARCHING

The search algorithm of a *PR*-tree is based on the algorithm to check whether two parametric rectangles intersect at any time instance. Given two parametric rectangles $r1$ and $r2$, let $\Pi_{i,t}(r1)$ and $\Pi_{i,t}(r2)$ denote the projection of $r1$ and $r2$ on the (x_i, t) space, then $r1$ and $r2$ intersect if and only if there is at least one time instance t_1 such that for each $i = 1, \ldots, d$, $\Pi_{i,t}(r1)$ and $\Pi_{i,t}(r2)$ intersect at t_1.

To determine if such a t_1 exists, we first compute for each dimension x_i, the time interval when $\Pi_{i,t}(r1)$ and $\Pi_{i,t}(r2)$ intersect. Then we check whether the intersection of the d time intervals is non-empty. If it is not empty, then the two parametric rectangles intersect, otherwise they do not intersect.

THEOREM 4.1. *Whether two d-dimensional parametric rectangles intersect can be checked in $O(d)$ time.*

Proof: The proof follows from the correctness of the algorithm outlined in Figure 5. □

5. INSERTION AND DELETION

5.1 Insertion

The insertion algorithm of *PR*-trees is an extension of the insertion algorithm of *R*-trees. We go down the tree to find an appropriate leaf node to insert the new index record, split the nodes that overflow, then propagate the split and the change of the bounding box upward. However, new criteria for choosing the appropriate subtree and splitting need to be designed for *PR*-trees.

Appropriate Subtree: At each level we choose the child whose bounding parametric rectangle needs least volume enlargement to include the new tuple. More precisely, let $r = (x_1^{\lfloor}, x_1^{\rfloor}, \ldots, x_d^{\lfloor}, x_d^{\rfloor},\ t^{\lfloor}, t^{\rfloor})$ be a d-dimensional parametric rectangle and P be the corresponding polyhedron in the $(x_1, \ldots, x_d,\ t)$ space. We define the *volume of* r, denoted by

Input: Two parametric rectangles ($r1$, $r2$)
Output: 1 - intersect, 0 - not intersect
1. $t^{\lfloor} = \max(r1.t^{\lfloor},\ r2.t^{\lfloor})$
2. $t^{\rfloor} = \min(r1.t^{\rfloor},\ r2.t^{\rfloor})$
3. **FOR** each dimension x_i **DO**
4. **IF** $r1.x_i^{\lfloor}$ is parallel to $r2.x_i^{\rfloor}$ **AND** $r1.x_i^{\lfloor}(t^{\lfloor}) > r2.x_i^{\rfloor}(t^{\lfloor})$ **THEN**
5. **RETURN** 0
6. **IF** $r1.x_i^{\lfloor}$ is not parallel to $r2.x_i^{\rfloor}$ **THEN**
7. Determine the time t' when $r1.x_i^{\lfloor}(t') = r2.x_i^{\rfloor}(t')$;
8. **IF** the slope of $r1.x_i^{\lfloor}$ is greater than the slope of $r2.x_i^{\rfloor}$ **THEN**
9. $t^{\rfloor} = \min(t', t^{\rfloor})$
10. **ELSE** $t^{\lfloor} = \max(t', t^{\lfloor})$
11. **IF** $r1.x_i^{\rfloor}$ is parallel to $r2.x_i^{\lfloor}$ **AND** $r1.x_i^{\rfloor}(t^{\lfloor}) < r2.x_i^{\lfloor}(t^{\lfloor})$ **THEN**
12. **RETURN** 0
13. **IF** $r1.x_i^{\rfloor}$ is not parallel to $r2.x_i^{\lfloor}$ THEN
14. Determine the time t'' when $r1.x_i^{\rfloor}(t'') = r2.x_i^{\lfloor}(t'')$;
15. **IF** the slope of $r1.x_i^{\rfloor}$ is greater than the slope of $r2.x_i^{\lfloor}$ **THEN**
16. $t^{\lfloor} = \max(t'', t^{\lfloor})$
17. **ELSE** $t^{\rfloor} = \min(t'', t^{\rfloor})$
18. **END-FOR**.
19. **IF** $t^{\lfloor} \le t^{\rfloor}$ **RETURN** 1
20. **ELSE RETURN** 0

Figure 5: Algorithm for Intersection Check

$vol(r)$, to be the volume of P. Since for each time instance t' between $t^{\lfloor}$ and $t^{\rfloor}$, the intersection of P with the plane $t = t'$ is a rectangle with area $\prod_{i=1}^{d}(x_i^{\rfloor} - x_i^{\lfloor})(t')$, the volume of r is the integral of the area function as follows:

$$vol(r) \quad = \quad \int_{t^{\lfloor}}^{t^{\rfloor}} \prod_{i=1}^{d}(x_i^{\rfloor} - x_i^{\lfloor})\ dt \qquad (1)$$

Since the lower bound ($x_i^{\lfloor}$) and the upper bound ($x_i^{\rfloor}$) of r are linear functions of t, the area function is a d-degree polynomial function of t, which can be translated into the following form:

$$a_0 + a_1 t + a_2 t^2 + \ldots + a_d t^d$$

where $a_0, a_1, a_2, \ldots, a_d$ are constants. Thus we can compute $vol(r)$ by translating Formula 1 to Formula 2:

$$vol(r) \quad = \quad (a_0 t + \frac{a_1}{2}t^2 + \frac{a_2}{3} + \ldots + \frac{a_d}{d+1}t^{d+1})\Big|_{t^{\lfloor}}^{t^{\rfloor}} \qquad (2)$$

Let the children of a non-leaf node be $E_1, \ldots, E_p$ and r_j be the bounding parametric rectangle of E_j. Suppose we would like to insert a new tuple T into one child of the node. Let $MBPR(r_j,\ T)$ denote the minimum bounding parametric rectangle of r_j and T. We will choose the child that needs the minimum the enlargement to include T, that is

$$\min_j\ enlarge(E_j, T)$$

where $enlarge(r_j,\ T) = vol(MBPR(r_j, T)) - vol(r_j)$.

EXAMPLE 5.1. *Let us consider the* PR*-tree in Example 3.1 again. Suppose we would like to add a new tuple* $r12$*, where*

$r12$*:*

$x^{\lfloor}$	$x^{\rfloor}$	$y^{\lfloor}$	$y^{\rfloor}$	$t^{\lfloor}$	$t^{\rfloor}$
$70 + 4t$	$90 + 3t$	$45 + 6t$	$55 + 6t$	*2*	*10*

We start at the root of the PR*-tree and try to add* $r12$ *to* $r1$, $r2$ *or* $r3$*. First we compute* $enlarge(r1,\ r12)$*. Figure 6(left) shows that the projection of* $r1$ *in* (x, t) *space already includes that of* $r12$*. So we do not need to enlarge* $r1$ *in the* x *dimension. Figure 6(right) shows the projection of* $r1$ *and* $r12$ *on* (y, t) *space. The dashed line segments* AB *and* BC *are the lower half of the convex hull of* $r1$ *and* $r12$ *in* (y, t) *space. The lower bound of* $MBPR(r1, r12)$ *in* y *dimension is the extension of* BC *and the upper bound of* $MBPR(r1, r12)$ *in* y *dimension is that of* $r1$*. Therefore MBPR(r1, r12) is*

$$(30 + 7t, 90 + 7t, 46.25 + 5.375t, 100 + 6t, 0, 10)$$

and

$$\begin{aligned} enlarge(r1, r12) &= vol(MBPR(r1, r12) - vol(r1) \\ &= \int_0^{10} 60(53.25 + 0.625t)dt \\ &\quad - \int_0^{10} 60(50 + t)dt \\ &= 825 \end{aligned}$$

Similarly, we compute $enlarge(r2,\ r12)$ *and* $enlarge(r3,\ r12)$ *as following Obviously,* $r1$ *is the best choice.*

$$\begin{aligned} MBPR(r2, r12) &= (0, 90 + 4t, 10 + 5t, 55 + 6t, 0, 10) \\ enlarge(r2, r12) &= 20683 \\ MBPR(r3, r12) &= (60 - 6t, 100 + 2t, 0, 55 + 6t, 0, 10) \\ enlarge(r3, r12) &= 51667 \end{aligned}$$

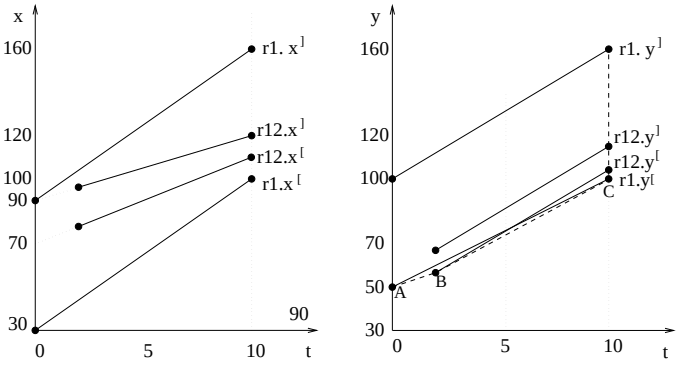

Figure 6: Enlarge $r1$ **to include** $r12$

Node Splitting: When a new entry is added to a full node with M entries, it is necessary to divide the collection of $M+1$ entries between two nodes. The division should be done in a way that makes the search as efficient as possible.

The *PR*-tree node splitting algorithm is an extension of the quadratic split algorithm of *R*-tree [11]. The idea is to choose two of the $M+1$ entries whose minimum bounding parametric rectangle has the largest volume increase as the first elements of the two new groups, where the volume increase is the volume of their MBPR minus their volume. Each of the remaining parametric rectangle is inserted into the group that needs less volume enlargement to include it.

EXAMPLE 5.2. *Let us continue Example 5.1. Suppose that the maximum number of children of each node is 3 and the minimum number of children is 2. We need to split the node $r1$ since it contains 4 children after the insertion of $r12$ as shown in Figure 7(1). For each pair of children, we compute the volume increase of their minimum bounding parametric rectangles as shown in the following table, since the matrix is symmetric, we only fill the elements in the upper triangle:*

	$r5$	$r6$	$r12$
$r4$	*16833*	*8250*	*16500*
$r5$	-	*14750*	*5656.25*
$r6$	-	-	*9556*

Since the MBPR of $r4$ and $r5$ has the largest volume increase, we first put them in two new nodes $r13$ and $r14$, respectively. Then we add $r12$ to $r14$ since $enlarge(r14, r12) < enlarge(r13, r12)$. We remove $r1$ and insert $r13$ and $r14$ into root as shown in Figure 7(2). This causes the splitting of the root and the height of the tree increases by 1 as shown in Figure 7(3). The process of the splitting of the root is similar to the splitting of $r1$.

THEOREM 5.1. *The insertion of a PR-tree can be done in $O(\log_M N)$ time, where M is the page size (maximum number of children per node) and N is the number of moving objects.*

Proof: It is easy to see that the height of the PR-tree for N moving objects is $\log_M N$. At each level, it took $O(M) = O(1)$ time to choose an appropriate child. Hence to choose the leaf to which the new entry is inserted can be done in $O(\log_M N)$ time.

If there is no node splitting, by Theorem 3.1, the MBPR of a node of M entries can be computed in $O(M \log M) = O(1)$ time, hence to propagate the change of MBPR upward can be done in $O(\log_M N) \times O(1) = O(\log_M N)$ time. If there are node splittings, it is easy to see that to split a node of M entries takes $O(M^2) = O(1)$ time, there are $O(\log_M N)$ nodes that need to be splitted. Therefore, node splitting can be done in $O(\log_M N)$ time. Therefore, to insert a new entry can be done in $O(\log_M N + \log_M N) = O(\log_M N)$ time. □

5.2 Deletion

To delete a moving object, we first need to find the leaf node in a *PR*-tree that contains the index record E of the moving object. This is different from the PR-tree searching algorithm for intersection queries in that in a non-leaf node we choose the subtree whose MBPR includes E instead of

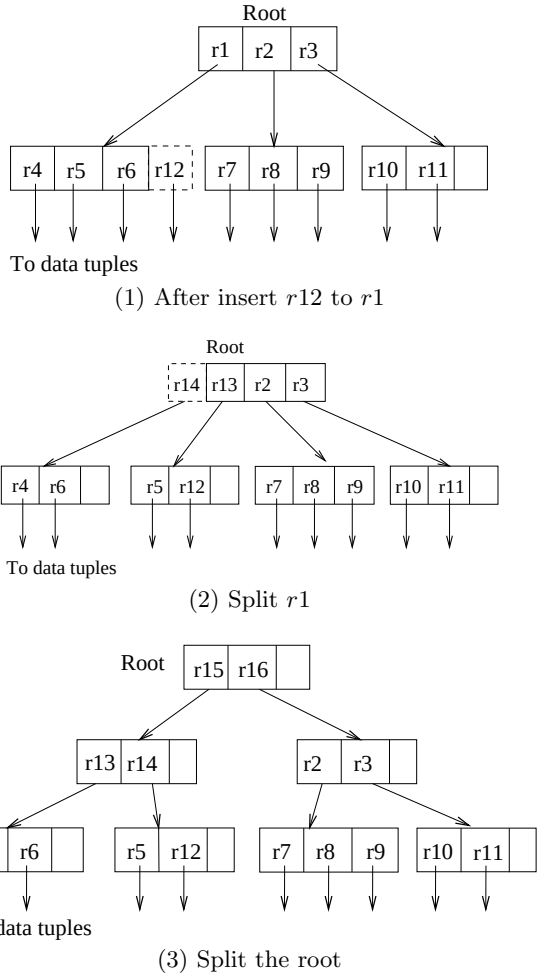

Figure 7: Procedure of inserting $r12$

intersect with E. Let $r1$ and $r2$ be two parametric rectangles, then we can check whether $r1$ *includes* $r2$ by verifying that the following conditions hold:

1. $r1.t^{\lfloor} \leq r2.t^{\lfloor}$ and $r1.t^{\rfloor} \geq r2.t^{\rfloor}$
2. Let $t1 = r2.t^{\lfloor}$ and $t2 = r2.t^{\rfloor}$, for each dimension x_i, $i = 1, \ldots, d$, $r1.x_i^{\lfloor}(t1) \leq r2.x_i^{\lfloor}(t1)$ and $r1.x_i^{\lfloor}(t2) \geq r2.x_i^{\lfloor}(t2)$.

After we delete E from a leaf node, if there are at least $M/2$ remaining entries, then we tighten the MBPR for the node and propagate this upward. If there are less than $M/2$ remaining entries in the node, we eliminate the node, use a sequence of re-insertions to relocate its entries and propagate the elimination upward if its parent has less than $M/2$ remaining entries. Re-insertion is a modification of the insertion algorithm described in the previous subsection in that an entry from a non-leaf node need to be re-insert to a non-leaf node at the same level as it used to be.

THEOREM 5.2. *To delete an index record E from a PR-tree can be done in $O(\log_M^2 N)$ time, where M is the page*

size (max number of children per node) and N is the number of moving objects.

Proof: Let L be the leaf that contains E. If the number of remaining entries in L is greater than or equal to $M/2$ after E is deleted, it is easy to see that deletion can be done in $O(\log_M N)$ time because for fixed M, the MBPR of a node can be recomputed in $O(M \log_2 M) = O(1)$ time,

If the number of remaining index records is less than $M/2$, then we need to delete the node and reinsert all the remaining entries. And in the worst case, this may propagate upwards to the root. Hence at most $O(\frac{M}{2} \times \log_M N) = O(\log_M N)$ entries may be reinserted. By Theorem 5.1, each entry can be reinserted in $O(\log_M N)$ time. Therefore each deletion takes $O(\log_M^2 N)$ in the worst case. □

6. A PERFORMANCE STUDY

We compared by experiments the performance of the *PR*-tree with an *R*-tree. We assumed that all objects are 2D moving rectangles, which are represented by parametric rectangles in the *PR*-tree and by 3D bounding boxes in the R-tree.

We assumed that in the *R*-tree node each entry contains a representation of a 3D rectangle by six numbers and a pointer. Assuming each number and the pointer is 4-bytes, the total size of an entry in the *R*-tree 28 bytes. Similarly, for an entry in the *PR*-tree we need 10 numbers (for each of $x^{\lfloor}, x^{\rfloor}, y^{\lfloor}, y^{\rfloor}$ the slope and the displacement values, and $t^{\lfloor}$ and $t^{\rfloor}$) plus a pointer, that is, a total size of an entry in the *PR*-tree is 44 bytes. We assumed a page size of 2048 bytes (the largest page size used in [11]) and the maximum number (M) of entries per node for the *R*-tree to be 73 and for the *PR*-tree to be 46, respectively.

We generated moving rectangles and represented them as $(x^{\lfloor},\ x^{\rfloor},\ y^{\lfloor},\ y^{\rfloor},\ t^{\lfloor},\ t^{\rfloor})$ where

$$t^{\lfloor} = 10k, \quad t^{\rfloor} = t^{\lfloor} + \Delta,$$
$$x^{\lfloor} = v_x t + a, \quad x^{\rfloor} = x^{\lfloor} + w,$$
$$y^{\lfloor} = v_y t + b, \quad y^{\rfloor} = y^{\lfloor} + l,$$
$$v_x = v\ cos(\theta), \quad v_x = v\ sin(\theta)$$

Each of the parameters $k, \Delta, a, b, w, l, v, \theta$ were randomly generated with a a uniform distribution. Parameter k was an integer between 0 and 100, Δ an integer between 80 and 120, a and b were rational numbers between 0 and 1000, the width w and the length l were rational numbers between 0 and 2, the speed v of the object was a rational number between 1 and 2, and the moving direction θ was an angle between 0 and 2π.

We varied N, the number of moving objects, from 32000 to 512000. For each set of data, we executed 100 randomly generated queries and computed the average number of I/Os per query. (Each of the queries was another moving rectangle generated similarly to the data set.) Figure 8 shows that the *PR*-tree needed fewer average number of I/Os per query than the R-tree.

We also did another set of experiments where we allowed the objects to shrink or grow in addition to moving. In this case, the lower and the upper bounds of a moving object in each dimension may have different speeds. That is, in this case we let $x^{\rfloor} = x^{\lfloor} + w + ct$ and $y^{\rfloor} = y^{\lfloor} + l + dt$, where c and d are uniformly distributed rational numbers over $[-\frac{w}{\Delta}, \frac{w}{\Delta}]$ and $[-\frac{l}{\Delta}, \frac{l}{\Delta}]$, respectively.[1] We can see that the *PR*-tree outperformed the R-tree in this case also Figure 9.

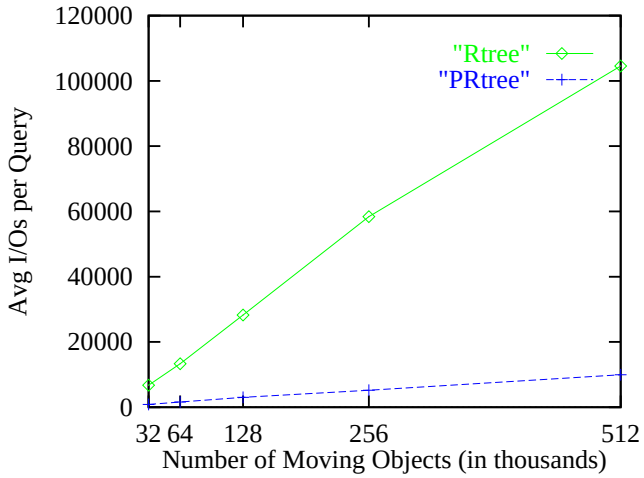

Figure 8: Performance for Fixed Size Moving Objects

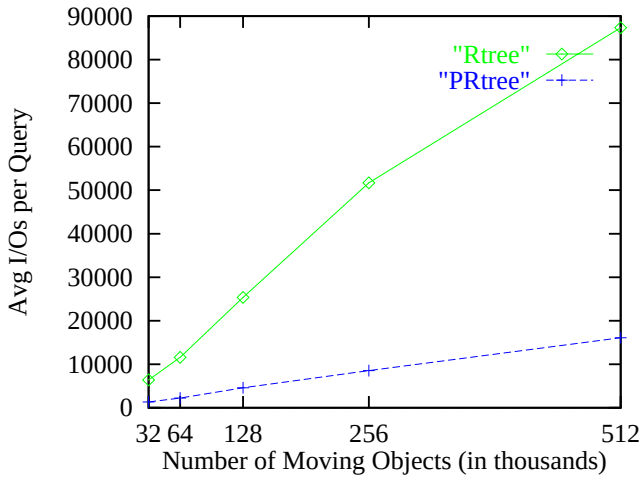

Figure 9: Performance For Growing or Shrinking Moving Objects

7. RELATED WORK AND CONCLUSION

Indexing moving objects is a novel area that is useful in a wide range of applications. Regular spatial indexes like R-trees [11] and its extensions R*-trees [2] and R+-trees [18] model stationary objects as high dimensional bounding boxes, but they are not efficient to index moving objects as shown in Section 6 since the minimum bounding rectangle assigned to a moving object requires much more space than necessary and therefor causes too many false hits.

Most current indexing approaches for moving objects are based on the moving point model [19] which represents mov-

[1]Note that choosing c and d in these ranges allow the rectangles to grow at most double size in each dimension or shrink to zero during its lifespan.

ing objects as a continuous function of time $f(t)$ and updates the database when the parameters of the motion like speed or direction change. It does not consider the spatial extents of the moving objects. [20] uses Quadtree [17] for indexing one dimensional moving points as line segment in the (x, t) plane. It partition the time dimension into time intervals of length H and indexes the part of the trajectory of each moving object that falls in the current time interval. This approach introduces substantial data replication in the index because a line segment is usually stored in several nodes.

[15] proposes to map the trajectories of a moving point represented by linear functions of the form $y = vt + a$ to a point (v, a) in the dual space and to index them by a regular spatial index structure such as a kd-tree [3]. [15] also provides theoretical lower bound on the number of I/Os to answer d-dimensional range searching problems. However, [15] assumes the trajectories of the moving objects extend to "infinity". It is not clear how to use this method to index two dimensional moving rectangles with finite duration.

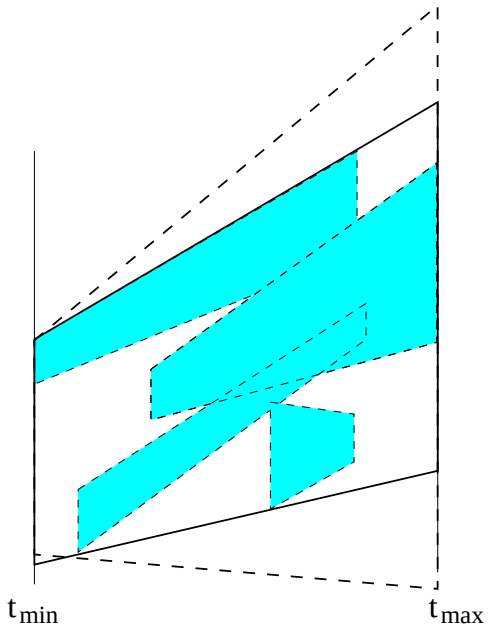

Figure 10: Different bounds of a set of objects

[16] proposes an R-tree like index structure for moving points which uses a time-parameterized "conservative" bounding rectangle to group moving points together. However, the "conservative" bounding rectangle of which the lower and the upper bounds is set to move with the minimum and the maximum speed of the enclosed points, respectively, is not a For example, tight bound of moving points. Figure 10 shows that the "conservative" bounding rectangle (dash line) of a set of moving objects is larger than the minimum bounding parametric rectangle of the objects that are used in PR-tree.

[1] proposes index structures for two dimensional moving points and shows that it can find all moving points lying in a static rectangle R in $O((N/B)^{1/2+\epsilon} + T/B)$ I/Os. However, it cannot be used to index moving objects with spatial extents. It does not consider the queries where R is moving.

Let N/B and T/N be the number of pages required to store N constraint tuples and T retrieved tuples, respectively. [4] describes a method that can answer half-plane queries for non-moving 2D linear constraint tuples in $O(log_B N/B + T/B)$ time assuming that the angular coefficient of the line associated with the half-plane query belongs to a fixed finite set. For moving objects no such upper bound is known yet.

8. REFERENCES

[1] P.K. Agarwal, L. Arge, and J. Erickson. Indexing mobile points. In *Proc. ACM Symposium on Principles of Database Systems*, pages 175–186, 2000.

[2] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proc. SIGMOD Intl. Conf. on Management of Data*, 1990.

[3] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18:9:509–517, 1975.

[4] E. Bertino, B. Catania, and B. Shidlovsky. Towards optimal two-dimensional indexing for constraint databases. *Information Processing Letters*, 64:1–8, 1997.

[5] A. Brodsky, J. Jaffar, and M. Maher. Towards practical query evaluation for constraint databases. *Constraints*, 2:3-4:279–304, 1997.

[6] M. Cai, D. Keshwani, and P.Z Revesz. Parametric rectangles: A model for querying and animating spatiotemporal databases. In *Proc. 7th International Conference on Extending Database Technology*, LNCS 1777, pages 430–444. Springer, 2000.

[7] M. Erwig, R.H. Güting, M.M. Schneider, and M. Vazirgiannis. Spatio-temporal data types: An approach to modeling and querying moving objects in databases. In *ACM Symposium on Geographic Information Systems*, 1998.

[8] M. I. Shamos F. P. Preparata. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

[9] L. Forlizzi, R.H. Güting, E. Nardelli, and M. Schneider. A data model and data structure for moving object databases. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 319–330, 2000.

[10] S. Grumbach, P. Rigaux, and L. Segoufin. The DEDALE System for Complex Spatial Queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 213–224, Seattle, Washington, USA, 1998.

[11] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. ACM-SIGMOD Conf. on Management of Data*, pages 47–57, 1984.

[12] P. Kanellakis, S. Ramaswamy, D.E. Vengroff, and J.S. Vitter. Indexing for data models with constraints and classes. In *Proc. ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 1993.

[13] P.C. Kanellakis, G.M. Kuper, and P.Z. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51:26–52, 1995.

[14] P. Kanjamala, P.Z. Revesz, and Y. Wang.

MLPQ/GIS: A GIS using linear constraint databases. In C. S. R. Prabhu, editor, *Proceedings of the 9th COMAD International Conference on Management of Data*, pages 389–393. Tata McGraw Hill, 1998.

[15] G. Kollios, D. Gunopulos, and V.J. Tsotras. On indexing mobile objects. In *Proc. ACM Symposium on Principles of Database Systems*, pages 261–272, 1999.

[16] S. Saltenis, C.S. Jensen, S.T. Leutenegger, and M.A. Lopez. Indexing the positions of continuously moving objects. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 331–342, 2000.

[17] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.

[18] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R^+-tree: A dynamic index for multi-dimensional objects. In *Proc. IEEE International Conf. on Very Large Databases*, pages 507–518, 1987.

[19] A.P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. In *Proceedings of the 13th IEEE International Conference on Data Engineering*, pages 422–432, 1997.

[20] J. Tayeb, O. Ulusoy, and O. Wolfson. A quadtree-based dynamic attribute indexing method. *The Computer Journal*, 41:3:185–200, 1998.