On R-trees with low query complexity

Mark de Berg

Joachim Gudmundsson

Mikael Hammar

Mark Overmars

UU-CS-2001-21

July, 2001

On R-trees with low query complexity

Mark de Berg[1], Joachim Gudmundsson[1], Mikael Hammar[2], and Mark Overmars[1]

[1] Department of Computer Science, Utrecht University,
PO Box 80.089, 35O8 TB Utrecht, the Netherlands.
{markdb, joachim, markov}@cs.uu.nl

[2] Department of Computer Science, Lund University,
Box 118, 221 00 Lund, Sweden. mikael@cs.lth.se

Abstract. The R-tree is a well-known bounding-volume hierarchy that is suitable for storing geometric data on secondary memory. Unfortunately, no good analysis of its query time exists. We describe a new algorithm to construct an R-tree for a set of planar objects that has provably good query complexity for point location queries and range queries with ranges of small width. For certain important special cases, our bounds are optimal. We also show how to update the structure dynamically, and we generalize our results to higher-dimensional spaces.

1 Introduction

Researchers in computational geometry have developed data structures for many types of queries on geometric data: point-location structures, range-searching structures, nearest-neighbor searching structures, and so on. The asymptotic worst-case behavior of these data structures is usually quite good—or at least close to the theoretical lower bounds. In practice, however, other kinds of data structures are often used. One reason is that in many applications storage is a very critical issue: $\Theta(n \log n)$ storage and even linear storage with a large constant factor can already be too much. Another reason is that the structures developed in computational geometry are usually dedicated to a very specific setting: a structure for searching with rectangular ranges in a set of line segments will not work for searching with rectangular ranges in a set of curve segments, or for searching with circular ranges in a set of line segments. In a typical application one needs to perform several different types of queries, and it is desirable to have a data structure that supports all, or at least many, of them.

An example of a versatile structure that is used in many applications is the bounding-volume hierarchy. This is a tree structure, whose leaves store the geometric data objects and whose internal nodes store a bounding box (or some other bounding volume) for the objects in the subtree rooted at that node. A bounding-volume hierarchy uses linear space and it can store any type of objects. It can perform range queries with any type of range; this means it can also do point location, since this is simply a range query with a point range.

The R-tree, which was proposed by Guttmann [7], is a bounding-volume hierarchy that is suitable for storing data on secondary storage. It can be considered a geometric version of a B-tree: all leaves are at the same depth, and all internal nodes, except for the root, have degree between t and $2t$, for a fixed parameter t which we call the *minimum degree* of the R-tree.[1] The root has a degree between 2 and $2t$. An internal node stores a bounding box for each of its subtrees; these bounding boxes are used to decide whether or not to visit a subtree when querying with a query range. The depth of an R-tree storing n objects in its leaves is $\Theta(\log n/\log t)$. The idea, like for B-trees, is to choose t as large as possible in order to minimize the depth of the tree, while making sure that each internal node still fits into one page of external memory. The R-tree is one of the most widely used geometric data structure in Geographic Information Systems—see for example the survey articles by Nievergelt and Widmayer [9] or by Six and Widmayer [11].

The key to the efficiency of an R-tree is how the underlying objects are grouped together in subtrees. Intuitively, for each subtree we would like the objects in its leaves to be clustered, so that their bounding box does not have too much empty space or overlap too many other bounding boxes. A number of heuristics has been proposed to achieve this [2, 3, 5–7, 10]. To our knowledge no construction algorithm has been described resulting in a structure with provably efficient worst-case performance. The only analytic result that we know of is by Faloutsos et al. [4]. Their setting is rather limited, however: they consider a 1-dimensional version of the R-tree, and assume that the input intervals have only one or two different sizes and that they are distributed uniformly. For this case they bound the number of nodes visited when answering a point-location query. They consider two heuristics to build the R-tree, and obtain bounds that are roughly $\Theta(\log n/\log t)$. Another result is by Becker et al. [1], who gives an optimal solution to a problem arising for some of the heuristics used to update an R-tree dynamically. The goal of our paper is to describe an algorithm for constructing R-trees whose worst-case query performance is good. We show this for point-location queries and for range queries with ranges of small width. Next we discuss our results in more detail.

Let $\mathcal{S}$ be a set of n objects for which we wish to construct an R-tree. A range query on $\mathcal{S}$ asks for all objects in $\mathcal{S}$ intersecting a query range Q. A point-location query is a range query where the query range is a point. Such queries are performed by traversing the tree starting at the root, visiting only subtrees whose bounding box is intersected by Q. The efficiency of the query procedure is determined by the number of nodes visited, since this number equals the number of disc accesses.

We define the *stabbing number* of a set of rectangles in the plane as the maximum number of rectangles stabbed by (that is, containing) any query point. For example, a set of disjoint rectangles has stabbing number equal to one. The worst-case number of nodes of the R-tree visited when answering a point-location

[1] The original definition allows between t and s rectangles for some given s with $s \geqslant 2t$, but for concreteness we assume $s = 2t$.

query corresponds to the stabbing number of the set of bounding boxes stored in the tree. The stabbing number of $\mathcal{R}_{\mathcal{S}}$, the set of bounding boxes of the objects in S, may already be n—take a set of n diagonal line segments that are very close together. Hence, we cannot achieve a sublinear bound on the number of visited nodes for general scenes. Therefore we will express our bounds in terms of σ, the stabbing number of $\mathcal{R}_{\mathcal{S}}$. A second parameter that we will use in our analysis is ρ, the *x-scale factor*, or *scale factor* for short, of $\mathcal{S}$. This is the ratio of the largest x-extent to the smallest x-extent of the objects in $\mathcal{S}$. (The x-extent of an object is the length of its projection onto the x-axis.) The scale factor has also been used by Zhou and Suri [12] for the analysis of a bounding-box heuristic, giving bounds on the number of intersections among the bounding boxes as compared to the number of intersections among the original objects.

We will prove that our construction algorithm produces an R-tree such that any point-location visits $O((\sigma + \lceil \log \rho \rceil) \log n / \log t)$ nodes. When σ and ρ are constant, which we expect to be true in many applications, this is optimal. In fact, our result is slightly more general than this—see the remark below Theorem 1. We can get rid of the dependency of ρ at the expense of an extra $O(\log n)$ factor, leading to an $O(\sigma \log^2 n / \log t)$ bound on the number of visited nodes. We also analyze the number of nodes visited by a range query. Here we obtain a bound of $O((\sigma + \lceil \log \rho \rceil + w + k) \log n / \log t)$, where w is the ratio of the x-extent of the query range to the smallest x-extent of any object in $\mathcal{S}$ and k is the number of reported objects.

Finally, we generalize our results to higher dimensions, and show how to update the R-tree dynamically.

2 The construction

Let $\mathcal{S}$ be a set of n disjoint objects in the plane, and let $\mathcal{R} = \mathcal{R}_{\mathcal{S}}$ be the set of bounding boxes of these objects. Let σ be the stabbing number of $\mathcal{R}$, that is, the maximum number of rectangles in $\mathcal{R}$ containing any query point. For convenience of presentation we shall sometimes pretend that $\mathcal{R}$ is the set for which we want to construct an R-tree. Of course, the R-tree for $\mathcal{R}$ is exactly the R-tree for $\mathcal{S}$. Let ρ denote the scale factor of $\mathcal{R}$ as defined above, which is equal to the scale factor of $\mathcal{S}$.

Before we proceed, let's give a more precise definition of the R-tree and of the terminology and notation that we will use. An R-tree for $\mathcal{S}$ is a tree $\mathcal{T}$ with the following properties.

- Each leaf node of $\mathcal{T}$, except when it is also the root, contains between t and $2t$ rectangles from $\mathcal{R}$. With each rectangle, a pointer to the corresponding object in $\mathcal{S}$ is stored.
- All leaves of $\mathcal{T}$ are at the same level.
- Each internal node ν of $\mathcal{T}$ stores for each of its subtrees the bounding box of all the rectangles stored in the leaves of that subtree.
  The bounding box of all bounding boxes stored at ν is denoted by $b(\nu)$. In other words, $b(\nu)$ is the bounding box of $\mathcal{R}(\nu)$, the set of rectangles stored in

the subtree rooted at ν. We say that $\mathcal{R}(\nu)$ is the *defining set* of $b(\nu)$. Notice that $b(\nu)$ is not stored at ν, but that it will be stored at the parent of ν.
– The root node of $\mathcal{T}$ has between 2 and $2t$ children, unless it is also a leaf. In the latter case it can contain between 1 and $2t$ rectangles from $\mathcal{R}$, with pointers to the corresponding objects in $\mathcal{S}$.

Sets with scale factor at most two. When the scale factor of $\mathcal{R}$ is two or less, we can proceed as follows. Assume without loss of generality that the smallest x-extent of any rectangle in $\mathcal{R}$ is equal to one. We partition the plane into vertical strips of unit width. We associate each rectangle in $\mathcal{R}$ with the strip containing its left edge, where strips are closed to the left and open to the right. A strip that has no rectangles associated with it is called *empty*, otherwise it is *non-empty*. Let $s_1, \ldots, s_k$ be the sequence of strips starting at the leftmost non-empty strip and ending at the rightmost non-empty strip. Notice that the sequence can contain empty strips—see Figure 1. Denote the set of rectangles associated to s_i by $\mathcal{R}(s_i)$, and let $n_i := |\mathcal{R}(s_i)|$. We number the rectangles in $\mathcal{R}$ in a left-to-right and bottom-to-top fashion, based on the strips: the rectangles associated to the leftmost strip are numbered $r_1, \ldots, r_{n_1}$ from bottom-to-top, the rectangles associated to the second leftmost non-empty strip are numbered $r_{n_1+1}, r_{n_1+2}, \ldots$ from bottom-to-top, and so on. We call the resulting ordering on the rectangles the *strip order*. Figure 1 illustrates it. The following observation will be crucial;

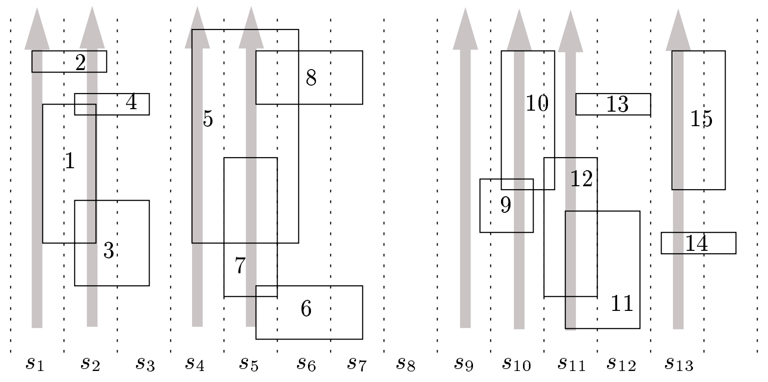


Fig. 1. The strip order.

it follows trivially from the fact that the x-extents of the rectangles are between one and two.

Observation 1 *Any rectangle intersecting a given strip s_i must be assigned to s_i, to s_{i-1}, or to s_{i-2}.*

The bounds (on the number of nodes visited by a query) we shall prove later apply to any R-tree that respects the strip order, that is, any R-tree for which the left-to-right order of the rectangles in the leaves corresponds to the strip order. For concreteness we will describe a simple bottom-up procedure to construct such an R-tree. An alternative way to construct the R-tree is by inserting the rectangles one at a time, using the insertion algorithm described in Section 4. Since the latter method already gives good behavior in terms of number of disc accesses, namely $\Theta(n \log n / \log t)$, we do not analyze the number of disc accesses for the bottom-up method.

The bottom-up construction works as follows. The bottom level of the R-tree consists of leaf nodes whose defining set have between t and $2t$ rectangles. This is achieved by simply letting the first (leftmost) leaf contain the first t rectangles in the strip order, the second leaf the next t rectangles, and so on. This continues until the number of remaining rectangles is at most $2t$, which are then put into the last (rightmost) leaf.

The level above the leaf level is built on the rectangles $b(\nu)$ for the leaf nodes ν, in the same way as the level before: the rectangles are put into groups of size t with the last group containing at most $2t$ rectangles. (Recall that the notation $b(\nu)$ is used to denote the bounding box of all rectangles in the subtree rooted at ν. Hence, for a leaf ν, we have that $b(\nu)$ denotes the bounding box of all rectangles stored in ν.) The ordering on the rectangles used to do the grouping is the left-to-right ordering of the leaves corresponding to the rectangles.

The remaining levels of the R-tree are constructed in the same way, always using the bounding boxes of the subtrees on the previous level. The process ends when the number of rectangles we are dealing with falls below $2t$. We then finish the R-tree by putting all rectangles into a single root node.

The method of constructing R-trees by first ordering the rectangles along a 1-dimensional curve has also been used by other authors [3, 10]. It has been observed that the main drawback of this method is that it disregards the sizes of the rectangles. Therefore we developed a new method, presented below, to deal with rectangles that differ a lot in size. Our analysis of the query complexity given in the next section is new as well.

The general case. So far we assumed that ρ, the scale factor of the set $\mathcal{R}$ of rectangles, is at most two. The algorithm we developed can also be used for larger ρ, but the dependency of the query complexity on ρ will be linear. We now describe a method that reduces the dependency to logarithmic.

The idea is to partition $\mathcal{R}$ into $m := \lceil \log \rho \rceil$ subsets, each with scale factor at most two. Let $\mathcal{R}_1, \ldots, \mathcal{R}_m$ be these subsets. For each $\mathcal{R}_i$ we can construct an R-tree with the algorithm described earlier. Since the depths of these R-trees may be different, however, we cannot simply merge them by constructing a tree on top of these R-trees. Another problem arising with this approach is that we may get internal nodes with too few children.

We therefore define a new ordering on the rectangles, as follows: rectangles are ordered by the index number of the set $\mathcal{R}_i$ they are in, and rectangles with the same index number are ordered using the strip order, as above. In other words, to obtain the sorted sequence of rectangles, we concatenate the sorted sequences for $\mathcal{R}_1, \ldots, \mathcal{R}_m$ in that order. We call the new order the *index-strip order.*

Now that we have a well-defined order on the rectangles, we can construct the R-tree as before (either using the bottom-up procedure, or the insertion algorithm described later).

Unbounded scale factors. When the scale factor gets really large, the method above gives rise to many subsets $\mathcal{R}_i$ and the resulting query complexity will not be very good (see below). We can overcome this problem with a simple trick: we replace the x-coordinate of the vertical edges of the rectangles by their rank. This way the x-'coordinates' that we are dealing with are integers between 1 and n, so the scale factor is bounded by n. We then apply our algorithm to these normalized rectangles. Conversion of the resulting R-tree to an R-tree for the original rectangles is trivial: simply replace the x-'coordinates' of the edges of the bounding boxes by the original coordinates. The latter step does not influence the query complexity. In the analysis given next, we can thus replace ρ by n if that gives a better result.

3 Analysis of the query complexity

Point-location queries. Suppose we perform a point-location query in the R-tree $\mathcal{T}$ with a point q. The number of nodes visited by the query procedure equals the number of bounding boxes stored in $\mathcal{T}$ stabbed by q.

Let ℓ_i denote the left bounding line of the strip s_i. We say that a bounding box b *straddles* ℓ_i if the defining set of b contains rectangles assigned to strips to the left of ℓ_i as well as rectangles assigned to strips to the right of ℓ_i. The following basic property of the construction will be important.

Lemma 1. *Let $\mathcal{T}$ be an R-tree constructed using the index-strip order for a set of rectangles with scale factor ρ. Let $\mathcal{B}_j(l)$ be the collection of bounding boxes of all nodes at a given level l in $\mathcal{T}$ with the property that the defining set of the bounding box has only rectangles from $\mathcal{R}_j$. For each line ℓ bounding a strip, the number of bounding boxes in $\mathcal{B}_j(l)$ straddling ℓ is at most one.*

Proof. By construction, the defining sets of the bounding boxes stored at level l form a disjoint partition of $\mathcal{R}$. Moreover, the left-to-right order of the defining sets of the nodes is consistent with the index-strip order. Consider all defining sets containing only rectangles from $\mathcal{R}_j$. Since we use the strip order within $\mathcal{R}_j$, there is at most one such defining set that has both a rectangle whose left edge is to the left of ℓ and a rectangle whose left edge is to the right of ℓ. $\square$

We can now prove a bound on the complexity of a point-location query.

Theorem 1. *Let $\mathcal{S}$ be a set of n objects in the plane such that the set of bounding boxes of $\mathcal{S}$ has stabbing number σ and scale factor ρ. For a given t, we can construct an R-tree of minimum degree t for $\mathcal{S}$ such that the number of nodes visited when answering a point-location query is $O((\sigma + \lceil \log \rho \rceil) \log n / \log t)$.*

Proof. Let l be a fixed level in the R-tree $\mathcal{T}$. Define $m := \lceil \log \rho \rceil$. We will show that the stabbing number of $\mathcal{B}(l)$, the set of bounding boxes of the nodes at level l, is at most $3\sigma + 7m - 1$.

Let q be a query point, and let s_i be the strip containing q. We consider three categories of bounding boxes in $\mathcal{B}(l)$ stabbed by q.

- *category (i): bounding boxes whose defining subset has rectangles from more than one of the subsets $\mathcal{R}_j$.*

  Because in the index-strip ordering rectangles with the same index are consecutive, there can be at most $m - 1$ such bounding boxes.

- *category (ii): bounding boxes not in category (i) that straddle ℓ_i, ℓ_{i-1}, or ℓ_{i-2}.*

  By Lemma 1 there are at most m such bounding boxes per bounding line (at most one for each subset $\mathcal{R}_j$), leading to at most $3m$ such bounding boxes in total.

- *category (iii): bounding boxes not in category (i) whose defining set contains only rectangles assigned to s_i, or only rectangles assigned to s_{i-1}, or only rectangles assigned to s_{i-2}.*

  Consider the bounding boxes whose defining set has only rectangles assigned to s_i. Such bounding boxes may have a defining set containing both a rectangle with bottom edge below q and one with bottom edge above q, as shown in Figure 2(a). Because of the ordering scheme within a strip, there are at most

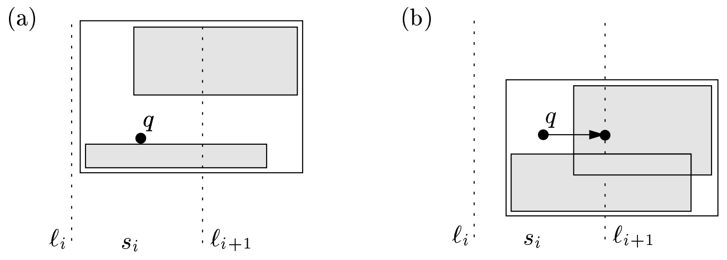


Fig. 2. Illustration for the proof of Theorem 1.

  m such bounding boxes (at most one per subset $\mathcal{R}_j$). Otherwise, the defining set of the bounding box has a rectangle $[x : x'] \times [y : y']$ with $q_y \in [y : y']$, as in Figure 2(b). Because the x-extent of each rectangle is at least the width of s_i, this means that such a rectangle must be stabbed by the orthogonal projection of q onto ℓ_{i+1}. There can be no more than σ such rectangles and, consequently, no more than σ such bounding boxes.

This shows that there are at most $\sigma + m$ bounding boxes stabbed by q whose defining set has only rectangles assigned to s_i. A similar argument works for bounding boxes stabbed by q whose defining set has only rectangles assigned to s_{i-1}, or to s_{i-2}. The only difference is that we now need to consider the projection of q onto ℓ_i, and onto ℓ_{i-1} respectively.

Adding up the bounds for each of the cases, we get a total bound of $3\sigma + 7m - 1$. Multiplying by the number of levels gives the desired bound. □

By applying the normalization described in the previous section, we can replace the factor ρ by n.

Corollary 1. *Let $\mathcal{S}$ be a set of n objects in the plane such that the set of bounding boxes of $\mathcal{S}$ has stabbing number σ and scale factor ρ. For a given t, we can construct an R-tree of minimum degree t for $\mathcal{S}$ such that the number of nodes visited when answering a point-location query is $O((\sigma + \log n) \log n / \log t)$.*

Remark. The only way in which the scale factor ρ plays a role in the proof of Theorem 1, is that it ensures that we can partition $\mathcal{R}$ into a logarithmic number of subsets with scale factor at most two. In general, our method gives a bound of $O(\sigma m \log n / \log t)$ for sets of rectangles that can be partitioned into m such subsets, even when $\log \rho$ is larger than m. For instance, if $\mathcal{R}$ contains three classes of rectangles—the large rectangles, the intermediate ones, and the small ones—each with scale factor at most two, then our method will work well even when the large rectangles are much larger than the small ones. Such a behavior may well occur for practical inputs.

Range-searching queries. Now suppose we want to perform a range query with an axis-parallel rectangular range Q. Let w denote the ratio of the x-extent of Q to the smallest x-extent of any object in $\mathcal{S}$. We call w the *width* of the range. Furthermore, let k denote the number of objects reported by the range query. We first analyze the number of nodes visited by the query procedure in terms of w, k, and the parameters introduced earlier. Then we show that in general—that is, for ranges that can be unbounded in both x- and y-direction—one cannot obtain similar (logarithmic) bounds.

Theorem 2. *The number of nodes visited when answering a range query with an axis-parallel rectangular range of width w is $O((\sigma + \lceil \log \rho \rceil + w + k) \log n / \log t)$.*

Proof. Define $m := \lceil \log \rho \rceil$. Let l be a fixed level in the R-tree $\mathcal{T}$, and let $\mathcal{B}(l)$ be the set of bounding boxes of the nodes at level l. We start by showing that the number of bounding boxes in $\mathcal{B}(l)$ intersecting the query range Q is $O(\sigma + m + w + k)$.

We consider five categories of bounding boxes in $\mathcal{B}(l)$ intersecting Q.

– *category (i): bounding boxes whose defining set has rectangles in more than one subset $\mathcal{R}_j$.*

  There are at most $m - 1$ such bounding boxes.

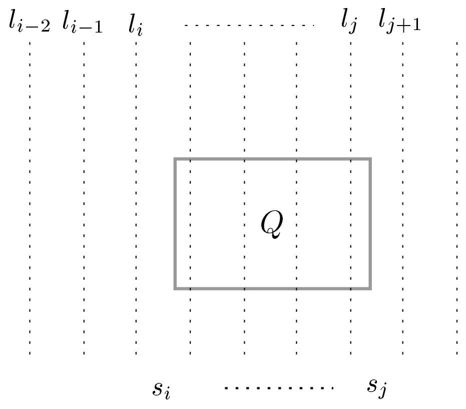


Fig. 3.

– *category (ii): bounding boxes not in category (i) containing a corner of Q.*

From the proof of Theorem 1, it follows that there are at most $3\sigma + 6m$ such bounding boxes per corner.

Fix a subset $\mathcal{R}_q$. Let $s_i, \ldots, s_j$ be the strips defined for $\mathcal{R}_q$ that are intersected by Q.

– *category (iii): bounding boxes not in category (i) straddling one of the lines $\ell_{i-2}, \ldots, \ell_{j+1}$.*

Note that $j - i \leqslant w_q + 1$, where w_q is the ratio of the width of Q and the width of the strips defined for $\mathcal{R}_q$. Illustrated in figure 3. By Lemma 1 there are at most $(j+1) - (i-2) \leqslant w_q + 4$ such bounding boxes for a subset $\mathcal{R}_q$. Since $w_q \leqslant w/2^q$, the total number of all bounding boxes of category (iii) is $\sum_{1 \leqslant q \leqslant m} w_q + 4 = O(w + m)$.

– *category (iv): bounding boxes not intersecting the top or bottom edge of Q and not in categories (i)–(iii).*

Such bounding boxes are either fully contained in Q or they intersect the left or right edge of Q. In the former case we can charge the intersection to one (in fact, many) object intersecting Q. In the latter case this is possible as well: Consider for example a bounding box b intersecting the left edge, e, of Q. Its defining set must have a rectangle whose right edge is to the right of e and a rectangle whose left edge is to the left of e. Let r_1 be a rectangle in the defining set of b whose left edge coincides with the left edge of b, and let r_2 be a rectangle in the defining set of b whose right edge coincides with the right edge of b. There are two cases. One is when e lies in the same strip as b belongs to. In this case r_1 must intersect e and, hence, we can charge the intersection to r_1, see figure 4(a). In the other case e must lie to the right

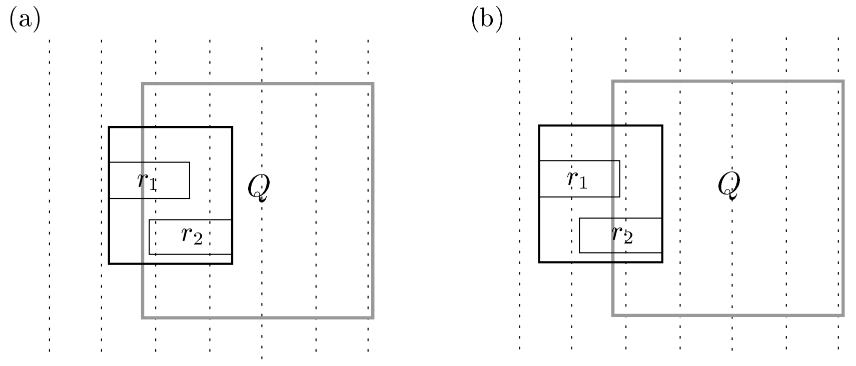


Fig. 4. Illustration for the proof of Theorem 2 (*category (iv)*).

of the strip that b belongs to, which implies that r_2 must either lie entirely within Q or intersect e, either way we may charge the intersection to r_2, as shown in figure 4(b). In the case when b only intersects the right edge e' of Q it holds that r_1 must lie entirely within Q or intersect e', thus, we may charge the intersection to r_1.
The total number of bounding boxes of category (iv) is $O(k)$.

– *category (v): bounding boxes intersecting the top or bottom edge of Q and not in categories (i)–(iii).*

  There are two cases. One is where the defining set of such a bounding box b has one rectangle whose bottom edge is below the bottom edge of Q and one rectangle whose bottom edge is above the top edge of Q. For each $\mathcal{R}_q$, this can happen only once for each of the strips defined for $\mathcal{R}_q$ and intersected by Q, or two strips to the left of Q. Hence, there are at most $w_q + 3$ such bounding boxes for $\mathcal{R}_q$, where w_q is defined as in case (iii), giving $O(w + m)$ such bounding boxes in total. In the other case the defining set of b must contain a rectangle whose top or bottom edge is contained fully in Q. This means that the object contained in this rectangle intersects Q. The total number of bounding boxes of category (v) is therefore $O(w + m + k)$.

Adding up the bounds for each of the cases, we get a total bound of $O(\sigma + m + w + k)$ for the number of nodes visited on a fixed level l. Over all levels we thus get a bound of $O((\sigma + m + w + k) \log n / \log t)$. □

Can we improve on this result? In particular, one would hope that it is possible to get rid of the dependence on w. Unfortunately, the next theorem shows that in this case one cannot get bounds close to the ones we just obtained.

Theorem 3. *For any n, there is a set $\mathcal{S}$ of n disjoint unit squares such that for any R-tree with minimum degree t on $\mathcal{S}$, there is a rectangular query range for* *which the query procedure will visit $\Omega(\sqrt{n}/\sqrt{t})$ nodes even though the range does not intersect any of the squares.*

Proof. Assume for simplicity that n is a perfect square. Consider a configuration of $\sqrt{n} \times \sqrt{n}$ disjoint squares arranged in a regular grid. The shaded squares in Figure 5 show the construction for $n = 16$. Let $\mathcal{T}$ be an R-tree for this collection

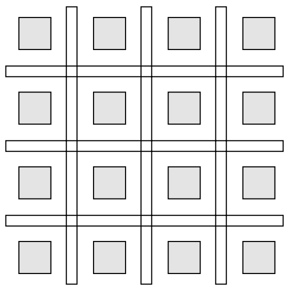


Fig. 5. The lower-bound configuration for range searching.

of squares. Consider the collection of $\Omega(\sqrt{n})$ long thin ranges separating either two consecutive columns or two consecutive rows of the set of squares. There are $\Omega(n/t)$ leaves in $\mathcal{T}$. It is easy to see that for any leaf, the bounding box of the squares stored at that leaf is intersected by $\Omega(\sqrt{t})$ ranges. Hence, the total number of range-box intersections is $\Omega(n/\sqrt{t})$, which implies that there must be a range intersecting $O(\sqrt{n}/\sqrt{t})$ bounding boxes. The number of leaves visited by the query procedure for this range is $O(\sqrt{n}/\sqrt{t})$. □

4 Dynamization

For the dynamic version, we assume that all coordinates are integers. Hence, the smallest x-extent that can ever occur is equal to one. To define the index-strip ordering we used in the previous section more formally, we define the following functions for a rectangle r:

$$\begin{aligned} \mathit{index\text{-}nbr}(r) &:= \lceil \log(x\text{-extent}(r)) \rceil \\ \mathit{strip\text{-}width}(r) &:= 2^{\mathit{index\text{-}nbr}(r)-1} \\ \mathit{strip\text{-}nbr}(r) &:= \lceil (x\text{-coordinate of left edge of } r)/\mathit{strip\text{-}width}(r) \rceil \\ \mathit{y\text{-}nbr}(r) &:= y\text{-coordinate of the bottom edge of } r \end{aligned}$$

We now define the following representation for r:

$$\mathit{rep}(r) := (\mathit{index\text{-}nbr}(r), \mathit{strip\text{-}nbr}(r), \mathit{y\text{-}nbr}(r)).$$

If we are working in the real RAM model we cannot use ceil/floor functions. However, we can easily compute *index-nbr*(r) for a given r in $O(\log \rho)$ time, where ρ is the scale factor. Similarly, we can compute *strip-nbr*(r) in $O(\log x_{\max})$ time, where $x_{\max}$ is the maximum x-coordinate that ever occurs.

Observation 2 *The ordering on the rectangles in $R = \{r_1, \ldots, r_n\}$ induced by a lexicographical ordering on the representations rep(r_i) is equal to the index-strip ordering as used in Section 2.*

This observation implies that if we augment the R-tree with some additional information, we can use standard leaf-oriented B-tree algorithms for insertions and deletions, as described in Chapter III.5.2 of Mehlhorns book [8]. The extra information is needed to be able to walk down the tree in order to locate the position of a new rectangle in the leaf-level of the R-tree. Recall that a bounding box b stored in an internal node v is the bounding box for the set of rectangles stored in the subtree of some child w_b of v. The extra information we need to store with b is the representation *rep*(r_b), where r_b is the minimum (according to the index-strip order) rectangle stored in the subtree of w_b. Now we can use the B-tree update algorithms, which require $O(\frac{\log n}{\log t})$ disk accesses per update. We obtain the following theorem.

Theorem 4. *The number of disc accesses for updates in the R-tree is $O(\frac{\log n}{\log t})$.*

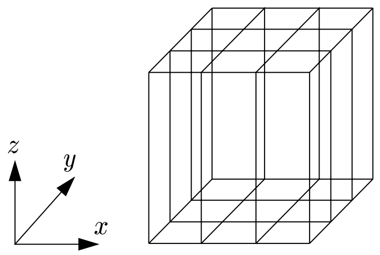
Remark. The extra information will force us to choose the minimum degree smaller, roughly by a factor of two, otherwise the information for a node would no longer fit into one page of external memory. This implies that the depth will increase by a factor of roughly $(1 + \log t)/\log t$.

It should also be noted is that the dynamization described above cannot be used together with the normalization trick described in Section 2. The normalization scales the input such that the x-coordinates of the corners of the rectangles are between 1 and n. This implies that when inserting a new elements into the R-tree the normalization will be affected. Some of the rectangles in $\mathcal{R}_i$ will then have to be moved to $\mathcal{R}_{i+1}$. The algorithm needs to remove the elements that change sets and then insert them into the R-tree again, which means that one cannot expect good worst-case update times.

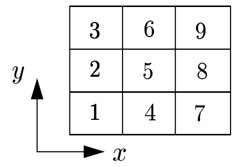
5 Higher-dimensional R-trees

The approach for the planar case extends easily into higher dimensions. For instance, suppose we have a set $\mathcal{S}$ of n objects in 3-dimensional space. As before, we let σ denote the stabbing number of the set $\mathcal{R}$ of bounding boxes of the objects in $\mathcal{S}$. We let ρ_x and ρ_y denote the x-scale factor and the y-scale factor of $\mathcal{R}$, respectively.

First assume that $\rho_x \leqslant 2$ and $\rho_y \leqslant 2$. We partition space into three-dimensional columns by planes orthogonal to the x-axis and planes orthogonal to the y-axis, as in Fig. 6(a). The spacing of the planes equals the minimum x-extent and y-extent, respectively, of the objects. We number the columns in increasing order primarily with respect to their x-coordinates and secondarily on their y-coordinates. Figure 6(b) shows an example of this in the projection. We assign each box in $\mathcal{R}$ to the column containing its front left edge (that is, the vertical edge with smallest x- and y-coordinate). We then number the boxes in $\mathcal{R}$ according to the ordering of the columns, where within each column we order the boxes based on the z-coordinate of their bottom facet. The latter ordering is done in increasing order. Given this new definition of the strip ordering, the



(a) partition of 3-space into strip columns



(b) ordering of the columns

Fig. 6. The ordering in 3-space.

construction proceeds in exactly the same way as in the planar case. Also the construction for sets where the scale factors are more than two is similar: partition $\mathcal{R}$ into $\lceil \log \rho_x \rceil \cdot \lceil \log \rho_y \rceil$ subsets with scale factors at most two, compute an order for each subset and concatenate the orders, and apply the standard construction algorithm. A similar approach works in dimensions higher than three. To describe this more precisely, we need to generalize the definition of the representation, given in the previous section, of a d-dimensional input box r. After that we will analyze the d-dimensional structures by showing the generalized results of Observation 1, Lemma 1 and Theorem 1.

Assume that all coordinates are integers in d-dimensional space $(x_1, \ldots, x_d)$. Hence, the smallest x_i-extent, $1 \leqslant i \leqslant d$, that can ever occur is equal to one. We define the following functions for a d-dimensional box r, where $1 \leqslant i < d$:

$$\begin{aligned} \textit{index-nbr}_i(r) &:= \lceil \log(x_i\text{-extent}(r)) \rceil \\ \textit{strip-width}_i(r) &:= 2^{\textit{index-nbr}_i(r)-1} \\ \textit{strip-nbr}_i(r) &:= \lceil (\text{the smallest } x_i\text{-coordinate of } r)/\textit{strip-width}_i(r) \rceil \\ x_d\textit{-nbr}(r) &:= \text{the smallest } x_d\text{-coordinate of } r \end{aligned}$$

We now define the following representation for r:

$$\begin{aligned} \textit{rep}(r) := (&\textit{index-nbr}_1(r), \ldots, \textit{index-nbr}_{d-1}(r), \\ &\textit{strip-nbr}_1(r), \ldots, \textit{strip-nbr}_{d-1}(r), x_d\textit{-nbr}(r)). \end{aligned}$$

Given a set of input boxes and the lexicographic ordering of the boxes defined by the above function, the R-tree is constructed exactly in the same way as a standard B-tree. The analysis of the number of nodes visited when answering a point-location query is very similar to the planar case.

For simplicity we will first consider sets where all the input boxes have the same index numbers, i.e. $\rho_{x_i} \leqslant 2$ for all $i < d$. Note that if the index numbers are the same for a set $\mathcal{R}_j$ then the strip widths will also be the same for all input boxes in $\mathcal{R}_j$. Since both the index numbers and the strip width are fixed we can now define a strip number for a single point q with respect to an input set $\mathcal{R}_j$ to be $\textit{strip-nbr}_i(q) := \lceil (x_i\text{-coordinate of } q)/\textit{strip-width}_i \rceil$.

Observation 3 *For every input box $r \in \mathcal{R}_j$ that is stabbed by a point q it holds that $\textit{strip-nbr}_i(q) - 2 \leqslant \textit{strip-nbr}_i(r) \leqslant \textit{strip-nbr}_i(q)$, for every $i < d$.*

The observation follows from the fact that the x_i-extent for all boxes are between one and two. The observation implies that the set of input boxes that may intersect q can be divided into at most 3^{d-1} sets, one set for each strip number.

A bounding box b is said to *straddle* a strip number SN if b contains two input boxes r_1 and r_2 of $\mathcal{R}_j$ such that $\textit{strip-nbr}(r_1) \leqslant \text{SN} < \textit{strip-nbr}(r_2)$.

The following basic property of the construction will be important.

Lemma 2. *Let $\mathcal{T}$ be an R-tree constructed using the index-strip order for a set of rectangles with scale factor $\rho_1, \ldots, \rho_{d-1}$. Let $\mathcal{B}_j(l)$ be the collection of bounding boxes of all nodes at a given level l in $\mathcal{T}$ with the property that the defining set of the bounding box has only rectangles from $\mathcal{R}_j$. For each strip number SN the number of bounding boxes in $\mathcal{B}_j(l)$ straddling SN is at most one.*

The proof is a straight-forward generalization of the proof of Lemma 1.

Note that the geometric interpretation of a strip number is a d-dimensional column, denoted a strip column, as in the two and three dimensional case. An input box r has strip number SN if and only if the point in r with lowest strip number has strip number SN.

Theorem 5. *Let $\mathcal{S}$ be a set of n objects in d-space such that the set of bounding boxes of $\mathcal{S}$ has stabbing number σ. Let ρ_{x_i} denote the x_i-scale factor of $\mathcal{R}$. For a given t, we can construct in $O(n \log n)$ time an R-tree of minimum degree t for $\mathcal{S}$ such that the number of nodes visited when answering a point-location query is $O((\sigma + \prod_{1 \leqslant i < d} \lceil \log \rho_{x_i} \rceil) 3^d \log n / \log t)$. Alternatively, we can obtain an R-tree where $O((3^d \sigma + \log^{d-1} n) \log n / \log t)$ nodes are visited in a point-location query.*

Proof. Let l be a fixed level in the R-tree $\mathcal{T}$. Define $m := \prod_{1 \leqslant i < d} \lceil \log \rho_{x_i} \rceil$. We will show that the stabbing number of $\mathcal{B}(l)$, the set of bounding boxes of the nodes at level l, is at most $(2 \cdot 3^{d-1} + 1)m + 3^{d-1}\sigma - 1$.

Let q be a query point. We consider three categories of bounding boxes in $\mathcal{B}(l)$ stabbed by q.

- *category (i): bounding boxes whose defining subset has boxes from more than one of the subsets $\mathcal{R}_j$.*

Because in the index-strip ordering rectangles with the same index are consecutive, there can be at most $m-1$ such bounding boxes.

– *category (ii): straddling bounding boxes not in category (i) stabbed by q.*

  By Observation 3 and Lemma 2 there are at most 3^{d-1} such bounding boxes for each subset $\mathcal{R}_j$ leading to at most $3^{d-1}m$ such bounding boxes in total.

– *category (iii): bounding boxes not in category (i) whose defining set all have the same strip number.*

  Consider one of the subsets $\mathcal{R}_j$ and let SN be the strip number of q. Now, consider the bounding boxes whose defining set has only boxes with *strip-number* SN. Such bounding boxes may have a defining set containing both an input box with minimum x_d-coordinate smaller than the x_d-coordinate of q and one with minimum x_d-coordinates greater than the x_d-coordinate of q, see Figure 2(a) for an illustration in two dimensions. Because of the ordering scheme within a set of boxes with the same strip number, there are at most one such bounding box of $\mathcal{R}_j$. Hence, at most m in total. Otherwise, the defining set of the bounding box has a box with x_d-coordinates in the range $[x : x']$ with $q_{x_d} \in [x : x']$, as in Figure 2(b). Because the x_i-extent of each box is at least the width of the strip column with *strip-number* SN along the x_i-axis, this means that such a box must be stabbed by the orthogonal projection of q onto an edge of the strip column. There can be no more than σ such rectangles and, consequently, no more than σ such bounding boxes in total.
  This shows that there are at most $\sigma + m$ bounding boxes stabbed by q whose defining set has only boxes with strip number SN. A similar argument works for bounding boxes stabbed by q whose defining set has only boxes assigned to any of the other strip boxes, and according to Observation 3 there are at most 3^{d-1}, such sets, hence, the total number of bounding boxes stabbed by q, not in category (i) and whose defining set all have the same strip number is $3^{d-1}(\sigma + m)$.

Adding up the bounds for each of the cases, we get a total bound of $3^{d-1}\sigma - 1 + (2 \cdot 3^{d-1} + 1)m$. Multiplying by the number of levels gives the desired bound.

When ρ_{x_i} is very large, we can again use normalization to improve the bounds. $\square$

We cannot obtain bounds for range searching that are similar to the planar case. The reason is that even when $\sigma = 1$ it can happen that a range with small width intersects many of the boxes in $\mathcal{R}$ without intersecting any of the corresponding objects in $\mathcal{S}$.

6 Concluding remarks

We have given an algorithm to construct R-trees for sets of n objects in the plane and in higher dimensional spaces. We analyzed the number of nodes visited when answering a point-location query in terms of n, and σ (the stabbing number of the initial bounding boxes), and ρ (the scale factor). When σ and ρ are constant, our results are optimal.

Our results might be improved in several ways. First of all, it would be interesting to reduce the dependency on σ in our bounds. Ideally, we would like to replace σ by $\lceil \sigma/t \rceil$. Another question is whether it is possible to improve the $O(\log^2 n/\log t)$ bound that we get for constant σ to $O(\log n/\log t)$.

It would also be nice to find another way to deal with scale factors larger than two. Our method of partitioning the set into subsets with scale factor two or less works fine in theory, but it is questionable whether it works well in practice.

Acknowledgement

We thank Frank van der Stappen for stimulating discussions during the early stages of this research.

References

1. B. Becker, P. Franciosa, S. Gschwind, T. Ohler, G. Thiemt, and P. Widmayer. Enclosing many boxes by an optimal pair of boxes. In *Proc. 9th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, LNCS 577, pages 475–486, 1992.
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. ACM-SIGMOD International Conference on Management of Data*, pages 322–331, 1990.
3. C. Faloutos and I. Kamel. *Packed R-trees using fractals.* Report CS-TR-3009, University of Maryland, College Park, 1992.
4. C. Faloutos, T. Sellis, and N. Roussopoulos. Analysis of object oriented spatial access methods. In *Proc. ACM-SIGMOD International Conference on Management of Data*, pages 426–439, 1987.
5. D.M. Gavrila. R-tree index optimization. In *Proc. 6th International Symposium on Spatial Data Handling*, pages 771–791, 1994.
6. D. Greene. An implementation and performance analysis of spatial data access methods. In *Proc. 5th International Conference on Data Engineering*, pages 606–615, 1989.
7. A. Guttmann. R-trees: a dynamic indexing structure for spatial searching. In *Proc. ACM-SIGMOD International Conference on Management of Data*, pages 47–57, 1984.
8. K. Mehlhorn. Data Structures and Algorithms 1: Sorting and Searching. *EATCS Monographs on Theoretical Computer Science*, vol. 1, Springer-Verlag, Heidelberg, Germany, 1984.
9. J. Nievergelt and P. Widmayer. Spatial data structures: concepts and design choices. In M. van Kreveld, J. Nievergelt, T. Roos, and P. Widmayer (eds.), *Algorithmic Foundations of Geographic Information Systems*, LNCS 1340, pages 153–198, 1997.
10. N. Roussopoulos and D. Leifer. Direct spatial search on pictorial databases with packed R-trees. In *Proc. ACM-SIGMOD International Conference on Management of Data*, 1985.

11. H.-W. Six and P. Widmayer. Spatial access structures for geometric databases. In B. Monien and T. Ottmann (eds.), *Data Structures and Efficient Algorithms*, LNCS 594, pages 214–232, 1992.
12. Y. Zhou and S. Suri. Analysis of a bounding box heuristic for object intersection. In *Proc. 10th Annual Symposium on Discrete Algorithms (SODA)*, 1999. To appear in *Journal of the ACM*.