Box-Trees and R-trees with Near-Optimal Query Time[*]

Pankaj K. Agarwal[†] Mark de Berg[‡] Joachim Gudmundsson[§] Mikael Hammar[§]
Herman J. Haverkort[‡]

Abstract

A box-tree is a bounding-box hierarchy that uses axis-aligned boxes as bounding volumes. The stabbing number of a box-tree with respect to a given type of query is the maximum number of nodes visited when answering such a query. We describe several new algorithms to construct box-trees with small stabbing number with respect to queries with axis-parallel boxes and with points. We also prove lower bounds on the worst-case stabbing number for box-trees, which show that our results are optimal or close to optimal. Finally, we present algorithms to convert box-trees to R-trees, resulting in R-trees with (almost) optimal stabbing number.

1 Introduction

Motivation and problem statement. Preprocessing a set S of geometric objects in $\mathbb{R}^d$ for answering *window* queries—report all objects of S that intersect a query d-rectangle[1]—is central to many applications and has been widely studied in several areas, including computational geometry, computer graphics, spatial databases, GIS, and robotics. In order to expedite and simplify the data structure, a window query is answered in two steps. In the first step, called the *filtering* step, each object is replaced by the smallest box containing the object and the query procedure computes the bounding boxes that intersect the query window. (Instead of boxes, other simple shapes such as spheres, ellipsoids, cylinders have also been used.) The second step, called the *refinement step*, extracts the actual objects among these bounding boxes that intersect the query window [19, 9]. A few recent results show that under certain reasonable assumptions on the input objects, the number of bounding boxes intersecting a query window is not much larger than the number of objects intersecting the window, which makes this approach quite attractive; see the paper by Zhou and Suri [20] and the references therein. There has been much work on the filtering step, and we also focus on this step. More precisely, we wish to preprocess a set S of n d-rectangles in $\mathbb{R}^d$ so that all rectangles of S intersecting a query d-rectangle can be reported efficiently. We will refer to this query as the *rectangle-intersection query.* A related query is the *rectangle-containment* query in which we want to report all rectangles in S that contain a query point.

A number of data structures with good provable bounds have been proposed for answering rectangle-intersection queries. Unfortunately they are of limited practical use, because the amount of storage they use is rather high: $O(n \log n)$ storage and even $O(n)$ storage with a large hidden constant are often unacceptable. Therefore in practice one usually uses simpler data structures. A commonly used structure for answering rectangle-intersection queries, rectangle-

[*]The work by P.A. is supported by Army Research Office MURI grant DAAH04-96-1-0013, by a Sloan fellowship, by NSF grants EIA–9870724, EIA–997287, and CCR–9732787, and by grant from the U.S.-Israeli Binational Science Foundation. The work by H.H. is supported by the Netherlands' Organization for Scientific Research (NWO).

[†]Center for Geometric Computing, Department of Computer Science, Box 90129, Duke University, Durham, NC 27708-0129, USA.

[‡]Institute of Information and Computing Sciences, Utrecht University, PO Box 80.089, 3508 TB Utrecht, the Netherlands.

[§]Department of Computer Science, Lund University, Box 118, 221 00 Lund, Sweden.

[1]A d-rectangle (also called a d-dimensional hyperrectangle) is the product of d intervals $[a_1, b_1] \times \cdots \times [a_d, b_d]$.

containment queries, and in fact many other types of queries is the *bounding-box hierarchy*, sometimes also called AABB-tree: this is a tree $\mathcal{T}$, in which each leaf is associated with a rectangle of S, and each interior node ν is associated with the smallest box B_ν enclosing all the rectangles stored at the leaves of the subtree rooted at ν. All the rectangles of S intersecting a query rectangle R are reported by traversing $\mathcal{T}$ in a top-down manner. Suppose the query procedure is visiting a node ν. If $B_\nu \cap R = \emptyset$, there is nothing to do. If $B_\nu \subseteq R$, then it reports all rectangles stored in subtree rooted at ν. Finally, if $B_\nu \cap R \neq \emptyset$ but $B_\nu \not\subseteq R$, it recursively visits the children of ν. We say that R *crosses* a node ν if $B_\nu \cap R \neq \emptyset$ and $B_\nu \not\subseteq R$. If the fan-out of $\mathcal{T}$ is bounded, then the query time is proportional to the number of nodes of $\mathcal{T}$ that R crosses plus the number of rectangles reported. We call the *stabbing number* of $\mathcal{T}$, denoted $\sigma(S, \mathcal{T})$, to be the maximum number of its nodes crossed by a rectangle. It is therefore desirable to construct a bounding-box hierarchy with small stabbing number.

In many applications, especially in the database applications, the set S is too large to fit in the main memory, therefore it is stored on disk. In that case, the main goal is to minimize the number of disk accesses needed to answer a window query, and the performance of an algorithm is analyzed under the standard external memory model [2]. This model assumes that each disk access transmits a contiguous block of t units of data in a single *input/output operation* (or *I/O*). The efficiency of a data structure is measured in terms of the amount of disk space it uses (measured in units of disk blocks), the number of I/Os required to answer a query, and the number of I/Os need to construct the data structure. In the context of bounding-box hierarchies, several schemes have been proposed that construct a tree as above but in which the fanout of each node depends on t. Some notable examples of external-memory bounding-box hierarchies are various variants of R-trees; see the survey paper [12]. We can still define the *crossing* nodes and the *stabbing number* as earlier, and one can argue that the number of I/Os needed to answer a query is proportional to the stabbing number plus the output size.

In this paper we study the problem of constructing bounding-box hierarchies, both in main and external memory, that have low stabbing number.

Previous results. As noted above several efficient data structures have been proposed for answering a rectangle-intersection query. For example, Chazelle [7] showed that a compressed range tree can be used to answer a d-dimensional rectangle-intersection query in time $O(\log^{d-1} n + k)$ using $O(n \log^{d-1}(n)/(\log\log n))$ space. This data structure is too complex to be practical even in $\mathbb{R}^2$. Since a d-rectangle in $\mathbb{R}^d$ can be mapped to a point in $\mathbb{R}^{2d}$, we can use a kd-tree to construct a data structure of $O(n)$ size that can answer a d-dimensional rectangle-intersection query in time $O(n^{1-1/2d} + k)$ [1]. A number of heuristics based on kd-trees have also been proposed to answer rectangle-intersection queries [1, 18]. Several papers [13, 15, 16] describe how to construct bounding-box hierarchies or other bounding-volume hierarchies (for example using spheres or k-DOPs as bounding volumes), but they do not obtain bounds on the worst-case query complexity.[2]

Some of the most widely used external-memory bounding-box hierarchies are the R-tree and its variants. An R-tree, originally introduced by Guttmann [14], is a B-tree, each of whose leaves is associated with an input rectangle. All leaves of an R-tree are at the same level, the degree of all internal nodes except of the root is between t and $2t$, for a given parameter t, and the degree of the root varies between 2 and $2t$. We will refer to t as the *minimum degree* of the tree. Although several methods have been proposed [12, 10, 11, 17] for ordering the input rectangles along the leaves—varying from simple heuristics to space filling curves—to minimize the stabbing number, none of them guarantee the worst-case performance. In the worst case, a linear number of bounding boxes might intersect a query rectangle even if it intersects only $O(1)$ input rectangles. The only analytical result is by Faloutsos *et al.* [11], but they prove bounds on the query time only in the 1-dimensional case when the input intervals are

[2]Barequet *et al.* [3] gave an algorithm to construct a bounding-box hierarchy in $\mathbb{R}^2$, and they claimed that if the rectangles in S are pairwise disjoint, then any point lies in $O(\log n)$ bounding boxes. But the argument presented in the paper has a technical problem.

uniformly distributed and have at most two different lengths. Recently, de Berg *et al.* [4] described an algorithm for construcing an R-tree on rectangles in $\mathbb{R}^2$ so that all k rectangles containing a query point be reported in $O((\sigma+\log\rho)\log n/\log t)$ I/Os. Here ρ is the ratio of the maximum and the minimum x-lengths of the input rectangles, and σ is the *point-stabbing number* of S, that is, σ is the maximum number of input rectangles containing any point in the plane. They also described another algorithm for constructing an R-tree that can answer a rectangle-intersection query in $O((\sigma+\log\rho+w+k)\log_t n)$ I/Os, where ρ and σ are as above and w is the ratio of the x-length of the query rectangle to the smallest x-length of an input rectangle.

Our results. In this paper we first describe several algorithms for constructing box-trees, and we prove lower bounds on the worst-case stabbing number of box-trees. Our first algorithm constructs a box-tree on a set of (possibly intersecting) d-rectangles in $\mathbb{R}^d$ so that a rectangle-intersection query can be answered in $O(n^{1-1/d}+k)$ time. Our lower bound shows that this is optimal.

In the plane we show that if the input is disjoint—more generally, if the point-stabbing number σ of the input is small—how to construct a box-tree that still has almost optimal query time for rectangle-intersection queries, but much better query times for point queries. More precisely, the time for rectangle-intersection queries is $O(\sqrt{n}\log n+\sqrt{\sigma}\log^2 n+k)$, and the time for point queries is $O(\sqrt{\sigma}\log^2 n+k)$. We also develop a box-tree with $O((\alpha+\sqrt{\sigma})\log^2 n+k)$ query time when the aspect ratio α of the query rectangles is small. One would hope that similar improvements are possible in higher dimensions. One of our lower-bound results shows that this is not possible: in dimensions $d \geqslant 3$, the $\Omega(n^{1-1/d}+k)$ lower bound on the query complexity holds even for hypercubes as query ranges, and any bounding-box hierarchy that achieves this query time cannot have a better worst-case query time for point queries, even when the input consists of disjoint 'almost-unit-hypercubes'.

Finally, we give general methods to convert box-trees with small stabbing number into R-trees with small stabbing number. When we apply these results to our boxtrees, we improve the result of de Berg *et al.* [4]: our query complexity does not depend on the parameter w (which makes their query complexity linear in the worst case), and it is linear in $\sqrt{\sigma}$ instead of σ. We also introduce the concept of *semi-R-trees*; these are similar to ordinary R-trees—the degree of each internal node, except for the root, is between t and $2t$ for some given parameter t—except that the leaves do not have to be at the same level. We give a general algorithm to convert a box-tree with small stabbing number into a semi-R-trees with small stabbing number; the stabbing number obtained here is better than that for R-trees. This leads to semi-R-trees with (almost) optimal stabbing number.

2 Lower Bounds

In this section we give lower bounds on the stabbing number of semi-R-trees of minimum degree t in various settings. Since semi-R-trees are more general than R-trees, the same bounds hold for R-trees. By choosing $t=2$, we obtain lower bounds for box-trees.

We start with a simple generalization of the 2-dimensional lower bound given by de Berg *et al.* [4]. It is obtained by putting n unit hypercubes in an $n^{1/d}\times\cdots\times n^{1/d}$ grid; the argument showing that this gives the stated bound is given in the full paper.

Theorem 2.1 *For any n and $d \geqslant 2$, there is a set of n disjoint unit hypercubes in $\mathbb{R}^d$ with the following property: for any semi-R-tree $\mathcal{T}$ of minimum degree t there is a query box not intersecting any box from S such that a query with that box visits $\Omega((n/t)^{1-1/d})$ nodes in $\mathcal{T}$.*

Next we describe a construction that proves lower bounds on rectangle-containment queries and that will also be useful for a number of other cases. For any $\varepsilon > 0$, we call a d-rectangle an *ε-hypercube* if the length of each edge is between 1 and $1+\varepsilon$. We fix a parameter $\mu \geq 1$ and construct a set $S=\{b_0,\ldots,b_{n-1}\}$ of n ε-hypercubes in $\mathbb{R}^d$. We also construct two sets of query points Q_1 and Q_2, called *primary* and *secondary* point sets, that lie in the common exterior of rectangles in S and have the following crucial property: For any semi-R-tree $\mathcal{T}$ on S with minimum degree t, either a point of Q_1 lies in at least μ bounding boxes of $\mathcal{T}$ or a point of Q_2 lies in $\Omega((n/t)/\mu^{1/(d-1)})$ bounding boxes of $\mathcal{T}$. We first decsribe the set S and then construct the point sets.

Let $n_1, \ldots, n_{2d}$ be the outward normals of a d-rectangle. We can pair these normals into d pairs $(n_{11}, n_{12}), \ldots, (n_{d1}, n_{d2})$ so that no pair contains opposite normals, that is, $n_{i1} \neq -n_{i2}$ for $1 \leq i \leq d$. Let h_i be the 2-plane spanned by the vectors n_{i1} and n_{i2} and containing the origin. Let b be a d-rectangle containing the origin. Since $n_{i1} \neq n_{i2}$, the facets f_{i1}, f_{i2} of b normal to n_{i1} and n_{i2}, respectively, share a $(d-2)$-face f_i, which is orthogonal to the 2-plane h_i. The intersection of f_i and h_i is a point c_i. Conversely, by specifying a point c_i on each h_i, $1 \leq i \leq d$, we can represent a unique d-rectangle in which c_i lies on the facets normal to n_{i1} and n_{i2}. We will therefore define each rectangle $b_j \in S$ by a d-tuple $(c_j^1, \ldots, c_j^d)$, where the facets of b_j whose outward normals are n_{i1} and n_{i2} pass through c_j^i. We next describe how to choose the points c_j^i, for $1 \leq i \leq d$ and $0 \leq j < n$.

For each 2-plane h_i, we choose a line ℓ_i of slope -1; the exact equation of ℓ_i will specified below. We will refer to h_1 as the *primary* plane, and every h_i with $i > 1$ will be called a *secondary plane*. Set $\hat{\mu} = \mu^{1/(d-1)}$. We place n points $p_0^1, \ldots, p_{n-1}^1$ on ℓ_1 (sorted along ℓ_1) and set $c_j^1 = p_j^1$ for every $0 \leq j < n$. For each $j > 1$, we place $\hat{\mu}$ points $p_0^i, \ldots, p_{\hat{\mu}-1}^i$ on ℓ_i and assign c_j^i to these points as follows. Let $\alpha(j) = (\alpha_0(j), \ldots, \alpha_{d-2}(j))$ be the representation of $j \bmod \mu$ in radix $\hat{\mu}$, that is, $\sum_{j=0}^{d-2} \alpha_j \hat{\mu}^j = (j \bmod \mu)$. We set $c_j^i = p_{\alpha_i(j)}^i$. Note that $n/\hat{\mu}$ points have the same value of c_j^i. Finally, we choose ℓ_i and the points on ℓ_i so that each b_j is an ε-hypercube. The easy details are omitted from this abstract.

Finally, we choose a set Q_1 of $n-1$ points on the primary plane h_1 and a set Q_2 of $(d-1)(\hat{\mu}-1)$ points on the secondary planes, as follows. Suppose h_1 is the x_1x_2-plane. For each $1 \leq j \leq n-1$, we choose the point $q_j = (x_1(p_{j-1}^1), x_2(p_j^1))$ and add it to Q_1. In other words, if we regard the points on ℓ_1 as a staircase, Q_1 is the set of concave corners of the staircase. In order to construct Q_2, we repeat the same step for each of the secondary planes, thus obtaining $\hat{\mu}-1$ points on each of them. These points will be on the boundary of some of the input boxes, but we can shift them a little to make them disjoint from all input boxes.

Lemma 2.2 *Let $\mathcal{T}$ be any semi-R-tree of minimum degree t on the set S constructed above. Then either there is a primary query point contained in $\Omega(\mu)$ bounding boxes stored in $\mathcal{T}$, or one of the secondary query points is contained in $\Omega(n/(t\mu^{1/(d-1)}))$ bounding boxes stored in $\mathcal{T}$.*

Proof: We prove the lemma for box-trees; the generalization to semi-R-trees is described in the full paper. Suppose that all primary query points are contained in less than $\mu/2$ bounding boxes stored in the interior nodes in $\mathcal{T}$. Then the number of incidences between these points and interior nodes' bounding boxes is at most $(n-1)\mu/2$. Since there are $n-1$ interior nodes in $\mathcal{T}$, they store at least $(n-1)/2$ bounding boxes that contain less than μ primary query points. Observe that a bounding box for input boxes $b_j, b_j' \in S$ contains $|j-j'|$ primary query points, because there are that many concave corners in the staircase between corners c_j^1 and $c_{j'}^1$. We conclude that there are at least $(n-1)/2$ bounding boxes that store boxes $b_j, b_{j'}$ (and perhaps some more boxes) with $|j-j'| < \mu$. But if $|j-j'| < \mu$ then $j \not\equiv j' \pmod{\mu}$, so $\alpha(b_j) \neq \alpha(b_j')$. This implies that there is at least one i with $2 \leqslant i \leqslant d$ such that $c_j^i \neq c_{j'}^i$. Hence, the bounding box storing $b_j, b_{j'}$ will contain one of the secondary query points. So in total we have at least $(n-1)/2$ incidences between secondary query points and bounding boxes, so one of the $(d-1)(\widehat{\mu}-1) = O(\mu^{1/(d-1)})$ secondary query points is contained in $\Omega(n/\mu^{1/(d-1)})$ bounding boxes. □

We can use this lemma to prove lower bounds for several settings. We start with a lower bound for point queries.

Theorem 2.3 *For any n, $d \geqslant 2$, and $\varepsilon > 0$, there is a set S of n ε-hypercubes in $\mathbb{R}^d$ with the following property: for any semi-R-tree $\mathcal{T}$ of minimum degree t there is a point not contained in any box from S such that a query with that point visits $\Omega((n/t)^{1-1/d})$ nodes in $\mathcal{T}$.*

Next, we modify the above construction so that the same bound can be achieved in $d \geq 3$ even if the input consists of a set of n disjoint ε-hypercubes and the queries are hypercubes. The idea is to use the construction above in $(d-1)$-dimensional space and use the remaining dimension to turn the intersecting $(d-1)$-dimensional boxes into disjoint d-dimensional boxes; details are in the full paper.

Theorem 2.4 *For any n, $d \geqslant 3$, and $\varepsilon > 0$, there is a set S of n disjoint ε-hypercubes in $\mathbb{R}^d$ with the following property: for any semi-R-tree $\mathcal{T}$ of minimum degree t there is a hypercube not intersecting any box from S such that a query with that hypercube visits $\Omega((n/t)^{1-1/d})$ nodes in $\mathcal{T}$.*

Finally, we observe that the proof of the preceding theorem actually shows that in higher dimensions any semi-R-tree with small (say, polylogarithmic) query complexity for points must have large (near-linear) query complexity for ranges. More precisely, it shows the following result.

Theorem 2.5 *For any n, $d \geqslant 3$, and $\varepsilon > 0$, there is a set S of n disjoint ε-hypercubes in $\mathbb{R}^d$ with the following property: for any semi-R-tree $\mathcal{T}$ of minimum degree t, if the number of nodes visited by any point query is μ, then there is a hypercube not intersecting any box from S such that a query with that hypercube visits $\Omega(n/(t\mu^{1/(d-1)}))$ nodes in $\mathcal{T}$.*

3 From kd-trees to Box-Trees

In this section we describe and analyze several methods to construct box-trees using kd-trees. For convenience we will allow nodes of degree up to $2d+2$—it is easy to convert these trees to binary trees without affecting the asymptotic bounds on the stabbing numbers.

3.1 The configuration-space approach

The basic method. Let S be a set of n arbitrary, possibly overlapping, d-rectangles in $\mathbb{R}^d$, which we call the *workspace*. As noted in Introduction, a d-rectangle $b = \prod_{i=1}^{d}[x_i^-(b), x_i^+(b)]$ can be represented by a point $(x_1^-(b), x_2^-(b), ..., x_d^-(b), x_1^+(b), x_2^+(b), ..., x_d^+(b))$ in $\mathbb{R}^{2d}$, which we call the *configuration space*. We build a $2d$-dimensional kd-tree on these tuples. To transform the kd-tree in configuration space into a box-tree in workspace, proceed as follows. Replace the representative tuple in each leaf by the corresponding input box. Then, going bottom-up, store in each internal node the bounding box of its children. We call the resulting box-tree a *configuration-space boxtree*, or *cs-box-tree* for short.

For the analysis of the range stabbing number in the resulting cs-box-tree, we need the following fact about kd-trees, given here without proof.

Lemma 3.1 *The number of cells at depth i in a d-dimensional kd-tree that intersect an axis-parallel f-flat ($0 \leqslant f \leqslant d$) is $O(2^{if/d})$.*

A kd-tree and, hence, our box-tree has the following property: the number of objects stored in the two subtrees of any given node differ by at most one. We call such trees *perfectly balanced.* That our box-tree is perfectly balanced will be advantageous when we will convert it to an R-tree. We can now analyse the range stabbing number in a cs-box-tree.

Lemma 3.2 *Let S be a set of n possibly intersecting boxes in the plane. There is a perfectly balanced box-tree for S such that the number of nodes at level i that are visited by a range query with an axis-aligned box is $O(2^{i(1-1/d)}+k)$, where k is the number of boxes in S intersecting the query range.*

Proof: Let $Q = \prod_{i=1}^{d}[x_i^-(Q), x_i^+(Q)]$ be a query range. We can restrict our attention to the interior nodes visited, since the number of visited leaves is at most one more. We distinguish two types of visited interior nodes ν. The first type is where at least one of the input boxes stored in the subtree of ν intersects Q. Obviously there are only $O(k)$ such nodes at a given level i. The second type is where all input boxes in the subtree of ν are disjoint from Q. Any input box disjoint from Q must be separated from Q by a hyperplane through a facet of Q. Clearly not all input boxes can be separated from Q by the same hyperplane, otherwise the bounding box of ν would not intersect Q and ν would not be visited. Hence, there are at least two such hyperplanes separating Q from an input box in the subtree of ν.

Assume w.l.o.g. that $x_i = x_i^-(Q)$ is one of these separating hyperplanes, and let b be the input box it separates from Q. Then we must have $x_i^+(b) < x_i^-(Q)$. But there must also be a box b' with $x_i^+(b') > x_i^-(Q)$, otherwise the bounding box of ν would not intersect Q. We can conclude that the points representing b and b' in the configuration space lie on opposite sides of the hyperplane $x_i^+ = x_i^-(Q)$. Now this implies that the hyperplane $x_i^+ = x_i^-(Q)$ intersects the cell in configuration space of the node in the kd-tree corresponding to ν.

We can apply the same argument to the second hyperplane separating Q from an input box (the hyperplane $x_j = x_j^+(Q)$, for example), to show that there is a hyperplane in configuration space with points on opposite sides ($x_j^- = x_j^+(Q)$ in the example).

We can conclude the following. Suppose Q visits a node ν of the second type. Then in configuration space there is a pair of hyperplanes, both of the form $x_i^+ = x_i^-(Q)$ or $x_i^- = x_i^+(Q)$ and both intersecting the cell in configuration space of the node in the kd-tree corresponding to ν. But then the cell must also be intersected by the $(2d-2)$-flat that is the intersection of these two hyperplanes. By Lemma 3.1 there are only $O(2^{i(2d-2)/2d}) = O(2^{i(1-1/d)})$ such nodes at level i. $\square$

This leads directly to the following theorem.

Theorem 3.3 *Let S be a set of n possibly intersecting boxes in the plane. There is a perfectly balanced box-tree for S such that the number of nodes visited by a range query with an axis-aligned box is $O(n^{1-1/d} + k \log n)$, where k is the number of boxes in S intersecting the query range.*

Improving the query time. We now show how to reduce the $O(k \log n)$ term in the query complexity to $O(k)$. The idea is the same as in a priority search tree [5]: input elements (boxes in our case) that have a high chance of being an answer are pushed to high levels in the tree. In our case this means that we store boxes that extend the farthest in some direction high in the tree. More precisely, the construction of the tree $\mathcal{T}$ for a set S of boxes in $\mathbb{R}^d$ is as follows.

If $|S| = 1$, then $\mathcal{T}$ consists of a single leaf node storing the input box in S. Otherwise we make a node ν storing the bounding box B_ν of all boxes in S, and we proceed as follows. For each of the $2d$ inner normals of the facets of B_ν, take the box from S that extends furthest in the direction of that normal. This results in a set S^* of at most $2d$ boxes. Each box in S^* is put in a so-called *priority leaf*, which is an immediate child of ν. If the set $S \setminus S^*$ of remaining boxes contains less than two boxes, then this box (if it exists) is put as a leaf child of ν. If two or more boxes remain, we split the set of boxes into two (almost) equal-sized subsets with a hyperplane in configuration space. This hyperplane is orthogonal to the cutting direction of the level we are working at; as before, the cutting directions of the levels cycle through the $2d$ directions in configuration space.

The subset of boxes whose representative points lie to one side of the cutting hyperplane are stored recursively in one subtree of ν. The subset of boxes whose representative points lie to the other side of the cutting hyperplane are stored recursively in another subtree of ν.

Next we analyze the query complexity of the tree resulting from this construction, which we call a *cs-priority-box-tree.* In our analysis we bound the number of visited nodes of a given weight, where the weight of a node is defined as the number of input boxes stored in its subtree. This will be useful when we convert this box-tree into a semi-R-tree.

Lemma 3.4 *The number of nodes of weight at least w visited by a query with a query box Q is $O((n/w)^{1-1/d} + k)$.*

Proof: Let $Q = \prod_{i=1}^d [x_i^-(Q), x_i^+(Q)]$. We can restrict our attention to the visited nodes of weight at least $2d$, as the total number of visited nodes is at most a constant times larger than this number. Let ν be such a visited node of weight at least $2d$. There are two cases.

The first case is where one of the priority leaves directly below ν stores a box intersecting Q. Clearly there are at most k such nodes. The second case is when all priority leaves directly below ν store boxes disjoint from Q. Thus each such box is separated from Q by a hyperplane through a facet of Q. We claim that not all boxes can be separated by the same hyperplane. Suppose for a contradiction that there is a facet f whose containing hyperplane separates all boxes of the priority leaves from Q. Then in particular it would separate the box that extends furthest in the direction of the inner normal of the facet f, contradicting that Q intersects the bounding box stored at ν. So we have two distinct hyperplanes through facets of Q separating a box in the subtree of ν from Q.

The box-tree that we have constructed basically corresponds to a kd-tree in configuration space, as before. The priority leaves make that the tree in configuration space is strictly speaking not a kd-tree, but it is easy to see that Lemma 3.1 still holds. Moreover, there is still a one-to-one correspondence between nodes of the box-tree and nodes of the kd-tree in configuration space. Hence, we can use the fact that there are two distinct hyperplanes through facets of Q separating a box in the subtree of ν from Q in the same way as in the proof of Lemma 3.2: it implies that there is a $(2d-2)$-flat in configuration space (defined by a pair of facets of Q) intersecting the cell in the kd-tree corresponding to ν. It follows that the total number of nodes ν to which the second case applies at a given level i is $O(2^{i(1-1/d)})$.

To finish the proof, observe that nodes at the lowermost $\lfloor \log(w/(2d)) \rfloor$ levels have weight less than w. Adding the bounds for the second case on the remaining levels, we get $\sum_{i=0}^{\lceil \log n \rceil - \lfloor \log(w/(2d)) \rfloor} O(2^{i(1-1/d)}) = O((n/w)^{1-1/d})$. □

The following theorem follows directly.

Theorem 3.5 *Let S be a set of n possibly intersecting boxes in $\mathbb{R}^d$. There is a box-tree for S such that the number of nodes that are visited by a range query with an axis-aligned box is $O(n^{1-1/d} + k)$, where k is the number of boxes in S intersecting the query range.*

3.2 The kd-interval-tree approach

The cs-box-tree of the previous section has optimal query complexity for point queries (and for range queries) if the input consists of arbitrary, intersecting boxes. Unfortunately, if the input boxes are disjoint then the query complexity for point queries does not improve. In this section we develop a different box-tree, the *kd-interval tree*, whose query complexity is much better if σ, the point-stabbing number of the input set S, is small. The query complexity for range queries increases only slightly. This approach only works in two dimensions; Theorem 2.5 states that a similar result in more than two dimensions cannot be obtained.

The basic idea behind kd-interval trees is again to use a kd-tree, but this time in the workspace (which is now the plane). Since the objects in the workspace are rectangles, not points, many of them may intersect the cutting line. These boxes are taken out and handled separately, like in an interval tree. To make kd-interval trees more efficient, we introduce priority leaves, like in the previous section.

The 1-dimensional case. First we describe how a set S of boxes that all intersect a given line ℓ are handled. With a slight abuse of terminology, we call a tree for this case a 1-dimensional kd-interval tree. If $|S| = 1$, then $\mathcal{T}$ consists of a single leaf node storing the input rectangle in S. Otherwise we make a node ν storing the bounding box B_ν of all rectangles in S, and we proceed as follows. For each of the 4 inner normals of the edges of B_ν, take the rectangle from S that extends furthest in the direction of that normal. This results in a set S^* of at most 4 rectangles. Each rectangle in S^* is put in a so-called *priority leaf*. Consider the set of intersections of the edges of the remaining rectangles with ℓ. Let p be the median of these intersection points. The rectangles in $S \setminus S^*$ containing p are stored in a subtree of ν that is a 2-dimensional cs-priority-box-tree as described in the previous section. The rectangles in $S \setminus S^*$ completely to one side of p are stored recursively in one subtree of ν. The rectangles in $S \setminus S^*$ completely to the other side of p are stored recursively in another subtree of ν.

We call the nodes in the main 1-dimensional kd-interval tree *1D-nodes*. Such a node corresponds to an interval on the defining line ℓ. We call the nodes of the 2-dimensional cs-priority-box-trees *cs-nodes*.

We start by analyzing the query complexity when we query with a segment on the line ℓ. The proof of the following lemma is omitted.

Lemma 3.6 *If we query a 1-dimensional kd-interval tree storing a set S of n rectangles with a line segment on the defining line ℓ, then we visit at most $O(\log n + k)$ nodes, where k is the number of rectangles to be*

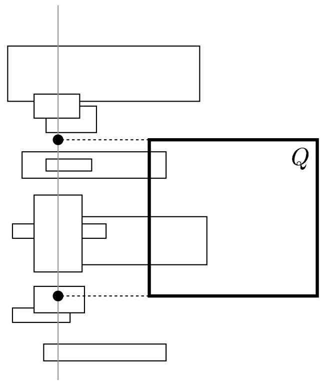

Figure 1: Querying a 1-dimensional kd-interval tree with a box Q.

reported.

Next we analyze the query complexity when we query with a box.

Lemma 3.7 *(i) If we query a 1-dimensional kd-interval tree storing a set S of n rectangles with a query box Q, then we visit at most $O(\sqrt{\sigma/w}\log n+k)$ nodes of weight at least w, where k is the number of rectangles to be reported.*
(ii) If the projection of Q onto the line ℓ that stabs the rectangles in S contains the intersections of all rectangles with ℓ, then the query time reduces to $O(k)$.

Proof: (i) See Figure 1. If Q intersects ℓ then the query is equivalent to querying with $Q \cap \ell$, so the result follows from the previous lemma. Otherwise, assume w.l.o.g. that ℓ is vertical and that Q lies to the right of ℓ. Consider a 1D-node ν that is visited when we query with Q. When the interval corresponding to this node is completely contained in the projection of Q onto ℓ, then the rectangle in the subtree extending furthest to the right must be intersected. This rectangle is stored in a priority leaf immediately below ν, to which we can charge the visit of ν. Hence, there cannot be more than k such nodes. When the interval is not completely contained in the projection of Q, then it contains an endpoint of the projection of Q, and there are only $O(\log n)$ such nodes.

Now consider a 2-dimensional cs-priority-box-tree that is visited. Suppose the interval of the 1D-node that is the parent of this subtree is completely contained in Q. Then we can argue again (using the priority leaves) that we can charge all the visited nodes to rectangles intersecting Q. If the interval of the 1D-node that is the parent of this subtree is not completely contained in the projection of Q, we argue as follows. First we observe that the interval must then contain an endpoint of the projection of Q, so there are only $O(\log n)$ such parent nodes. In the 2-dimensional configuration-space box-tree below such a parent, we can use Lemma 3.4 to bound the number of visited nodes of weight w by $O(\sqrt{n'/w} + k')$, where n' is the number of boxes stored in the configuration-space box-tree and k' is the number of answers reported in this subtree. Note that $n' = O(\sigma)$, since the configuration-space box-trees are used only to store sets of boxes that share a single point. Hence, the overal number of cs-nodes visited is $O(\sqrt{\sigma/w}\log n + k)$, finishing the proof of part (i) of the lemma.

(ii) The proof of the second part of the lemma can be found in the full version of this paper. □

The 2-dimensional case. Our kd-interval tree for a general set S of rectangles in the plane is defined as follows. If $|S| = 1$, then $\mathcal{T}$ consists of a single leaf node storing the input box in S. Otherwise we make a node ν storing the bounding box B_ν of all boxes in S, construct priority nodes (like in the 1-dimensional case) and split the remaining set of boxes by a vertical or horizontal line (depending on the level ν in the tree). This splitting line ℓ is chosen such that the number of rectangles lying completely to one side of ℓ differs at most one from the number lying completely to the other one side. The rectangles lying completely to one side of ℓ are stored recursively in one subtree of ν. The rectangles to the other side are stored recursively in another subtree of ν. The rectangles intersecting ℓ are stored in a 1-dimensional kd-interval tree, as explained above. We call the nodes of the main tree, which correspond to 2-dimensional cells, *2D-nodes*. Next we analyze the performance of the kd-interval tree.

Lemma 3.8 *The number of nodes of weight at least w that are visited by a range query with an axis-aligned box is $O(\sqrt{n/w}\log n + \sqrt{\sigma/w}\log^2 n + k)$, where k is the number of reported answers. The number of such nodes visited by a point query is $O(\sqrt{\sigma/w}\log^2 n + k)$.*

Proof: Consider a 2D-node that is visited when we query with an axis-aligned rectangle Q. We distinguish four different types of such nodes. We bound their number and the number of nodes visited in 1-dimensional kd-interval-subtrees separately.

Inner nodes: These are 2D-nodes whose bounding boxes lie completely inside Q. The number of inner nodes is easy to bound, since all rectangles in the subtree of such a node intersect Q. Hence, the total number of such nodes, or nodes in their 1-dimensional associated kd-interval trees, is $O(k)$.
Side nodes: These are 2D-nodes whose bounding boxes cut exactly one edge of Q. In this case the rectangle that extends furthest into the direction of the inner normal of this edge must intersect Q. This rectangle is stored in a priority leaf immediately below the node. The same reasoning applies to their 1-dimensional associated kd-interval trees. Hence, the total number of side nodes or nodes in their associated kd-interval trees is $O(k)$.
Piercing nodes: These are 2D-nodes that cut two opposing edges of Q, but do not contain any corners of Q. From Lemma 3.1 and the fact that nodes at the lowermost $\lfloor\log(w/(2d))\rfloor$ levels of the tree must have weight less than w, we conclude that the number of 2D-nodes of weight at least w that intersect any edge of Q is bounded by $\sum_{i=0}^{\lceil\log n\rceil - \lfloor\log(w/(2d))\rfloor} O(2^{i/2}) = O(\sqrt{n/w})$. Now there are two cases: the splitting line used at such a node ν is orthogonal to the intersected edges, or it is parallel to them. In the former case we can apply Lemma 3.6 to obtain a $O(\log n + k')$ bound on the number of nodes visited in the 1-dimensional kd-interval tree associated with ν, where k' is the number of reported answers. In the latter case we can apply Lemma 3.7(ii) to get a bound of $O(k')$. Hence, we get a grand total of $O(\sqrt{n/w}\log n + k)$.
corner nodes: These are 2D-nodes that contain one or more corners of Q. There are $O(\log n)$ such nodes. To obtain the total number of visited nodes in the associated 1-dimensional kd-interval trees, we have to multiply this by the bound of Lemma 3.7, leading to a total of $O(\sqrt{\sigma/w}\log^2 n + k)$.

There are no other types of nodes whose bounding boxes intersect Q. Adding up the number of nodes for all four cases gives the desired bound. $\square$

This leads to the following theorem.

Theorem 3.9 *Let S be a set of n possibly intersecting boxes in the plane, such that no single point is contained in more than σ boxes. There is a box-tree for S such that the number of nodes visited by a range query with an axis-aligned box is $O(\sqrt{n}\log n + \sqrt{\sigma}\log^2 n + k)$, where k is the number of boxes in S intersecting the query range. The number of nodes visited by a point query is $O(\sqrt{\sigma}\log^2 n + k)$.*

3.3 The longest-side-first approach

Recall that a kd-interval tree is basically a modified kd-tree, where each node is split by a line. The orientations of these lines depend on the level in the tree in such a way, that orientations take turns in a round-robin fashion on any path from the root down into the tree. An interesting variation of the kd-interval tree arises when we replace the round-robin splitting strategy by the longest-side splitting rule as suggested by Dickerson et al. [8]. In such a longest-side-first kd-tree, the number of nodes whose corresponding cell is pierced by a query rectangle is small if the query rectangle is fat. We can use this to prove the following result.

Theorem 3.10 *Let S be a set of n boxes in the plane with stabbing number σ. There is a box-tree for S such that the number of nodes that are visited by a range query with a rectangular range of aspect ratio α is $O((\alpha + \sqrt{\sigma})\log^2 + k)$, where k is the number of boxes in S intersecting the query range. The number of such nodes visited by a point query is $O(\sqrt{\sigma}\log^2 n + k)$.*

4 From box-trees to (semi-)R-trees

In the previous section we described several algorithms to construct box-trees with good query complexity. In this section we give general theorems to convert them to (semi-)R-trees.

We start with a general theorem that converts any box-tree to an R-tree. Recall that the *weight* of a box-tree node is the number of input boxes stored in its subtree.

Theorem 4.1 *Let $\mathcal{T}$ be a box-tree for a set of n boxes in $\mathbb{R}^d$ such that any query with a range of a given type visits at most $f(w)$ nodes of weight w or more. Then $\mathcal{T}$ can be converted to an R-tree of minimum degree t where every query with a range of the same type visits at most $O(f(t)\log n/\log t)$ nodes.*

Proof: We simply read out the leaves from $\mathcal{T}$ in order, and then construct an R-tree where the boxes occur in the same order in the leaves (for example, using an algorithm to construct B-trees).

Consider a bounding box B stored in the R-tree. It is the bounding box for some input boxes that were stored in consecutive leaves in the box-tree $\mathcal{T}$. Let $\nu(B)$ be the lowest common ancestor of these leaves. Since the minimum degree in the R-tree is t, the weight of $\nu(B)$ is t or more. Furthermore, the nodes $\nu(B)$ for the bounding boxes B stored at a fixed level in the R-tree must be distinct, because their defining sets form a partition of the leaves in $\mathcal{T}$ into consecutive sequences. Hence, we can charge the visited nodes of the R-tree to visited nodes of weight t or more in $\mathcal{T}$, in such a way that a node in $\mathcal{T}$ does not get charged more than once from nodes at a fixed level in the R-tree. Since the depth of the R-tree is $O(\log n/\log t)$, the bound follows. $\square$

The construction of Theorem 4.1 results in losing a logarithmic factor in the query complexity. In the full paper we show how to improve this result for perfectly balanced box-trees. Recall that a box-tree is called perfectly balanced if for any node the weight of its left and right child differ by at most one.

Theorem 4.2 *Let $\mathcal{T}$ be a perfectly balanced box-tree for a set of n boxes in $\mathbb{R}^d$ such that any query with a range of a given type visits at most $f(i)$ nodes at level i in $\mathcal{T}$. Then $\mathcal{T}$ can be converted to an R-tree of minimum degree t where every query with a range of the given type visits at most $O(\sum_{i=0}^{(\log n/\log t)-1} f(i\log t))$ nodes.*

Finally, we can show that that we can also improve Theorem 4.1 for the general case if we are willing to settle for semi-R-trees instead of real R-trees. (Recall that the difference between a semi-R-tree and an R-tree is that in the former we do not require all leaves to be at the same depth.)

Theorem 4.3 *Let $\mathcal{T}$ be a box-tree for a set of n boxes in $\mathbb{R}^d$ such that any query with a range of a given type visits at most $f(w)$ nodes of weight w or more. Then $\mathcal{T}$ can be converted to a semi-R-tree of minimum degree t where every query with a range of the same type visits at most $O(f(t))$ nodes.*

By applying the conversion algorithms of the theorems above to the structures from the previous section, we obtain the following results.

Corollary 4.4 *Let S be a set of n boxes in $\mathbb{R}^d$ with stabbing number σ.*
(i) There is an R-tree for S of minimum degree t such that the number of nodes visited by any box query is $O((n/t)^{1-1/d} + k\log n/\log t)$, where k is the number of reported answers.
(ii) There is an semi-R-tree for S of minimum degree t such that the number of nodes visited by any box query is $O((n/t)^{1-1/d} + k)$.
(iii) When $d = 2$, there is a semi-R-tree for S of minimum degree t such that the number of nodes visited by any box query is $O(\sqrt{n/t}\log n + \sqrt{\sigma/t}\log^2 n + k)$, and the the number of nodes visited by any point query is $O(\sqrt{\sigma/t}\log^2 n + k)$. In both bounds, k is the number of reported answers.
(iv) When $d = 2$, there is a semi-R-tree for S of minimum degree t such that the number of nodes visited by any query with a rectangle of aspect ratio α is $O((\alpha + \sqrt{\sigma/t})\log^2 n + k)$, where k is the number of reported answers.
(v) For the cases mentioned under (iii) and (iv) there is also an R-tree of minimum degree t for which the number of visited nodes is $O(\log n/\log t)$ times the number of visited nodes in the semi-R-tree.

References

[1] P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, volume 223 of *Contemporary Mathematics*, pages

1–56. American Mathematical Society, Providence, RI, 1999.

[2] A. Aggarwal and J. S. Vitter. The Input/Output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.

[3] G. Barequet, B. Chazelle, L.J. Guibas, J.S.B. Mitchell, and A. Tal. BOXTREE: A hierarchical representation for surfaces in 3D. *Computer Graphics Forum (CGF)* 15:C387–396, 1996.

[4] M. de Berg, J. Gudmundsson, M. Hammar, and M. Overmars. On R-trees with low stabbing number. *Proc. 8th European Symposium on Algorithms, LNCS 1879*, pages 167–178, 2000.

[5] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, Berlin, 1997.

[6] M. de Berg, M. Overmars, and O. Schwarzkopf. Computing and verifying depth orders. *SIAM J. Comput.* 23:437–446 (1994).

[7] B. Chazelle, A functional approach to data structures and its use in multidimensional searching, *SIAM J. Comput.*, 17 (1988), 427–462.

[8] M. Dickerson, C. Duncon, and M. Goodrich. K-D Trees Are Better when Cut on the Longest Side. *Proc. 8th European Symposium on Algorithms, LNCS 1879*, pages 179–190, 2000.

[9] T. Brinkhoff, H.-P. Kriegel, R. Schneider, and B. Seeger. Multi-step processing of spatial joins. In *Proc. ACM-SIGMOD International Conference on Management of Data*, 1994, 197–208.

[10] C. Faloutos and I. Kamel. *Packed R-trees using fractals.* Report CS-TR-3009, University of Maryland, College Park, 1992.

[11] 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.

[12] V. Gaede and O. Günther, Multidimensional access methods. *ACM Comput. Surveys* 30 (1998), 170–205.

[13] S. Gottschalk, M.C. Lin, and D. Manocha. OBB-Tree: a hierarchical structure for rapid interference detection. In *ACM Computer Graphics Proceedings*, pages 171–180, 1996.

[14] 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.

[15] P.M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. In *ACM transactions on graphics*, 15(3):179–210, 1996.

[16] J.T. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-DOPs. In *IEEE transactions on visulalization and computer graphics*, 4(1):21–36, 1998.

[17] S. Leutenegger, M. A. Lopez, and J. Edington, STR: A simple and efficient algorithm for R-tree packing, *Proc. 13th IEEE Internat. Conf. on Data Engineering*, 1997, pp. 497–506.

[18] 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.

[19] J. Orenstein, A comparison of spatial query processing techniques for native and parameter spaces, *Proc. ACM SIGMOD Conf. on Management of Data*, 1990, pp. 343–352.

[20] 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.*