Dynamic Orthogonal Range Searching on the RAM, Revisited*

Timothy M. Chan[1] and Konstantinos Tsakalidis[2]

1 **Dept. of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA**
tmc@illinois.edu

2 **Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada**
ktsakali@uwaterloo.ca

Abstract

We study a longstanding problem in computational geometry: 2-d dynamic orthogonal range reporting. We present a new data structure achieving $O\left(\frac{\log n}{\log\log n}+k\right)$ optimal query time and $O\left(\log^{2/3+o(1)} n\right)$ update time (amortized) in the word RAM model, where n is the number of data points and k is the output size. This is the first improvement in over 10 years of Mortensen's previous result [*SIAM J. Comput.*, 2006], which has $O\left(\log^{7/8+\varepsilon} n\right)$ update time for an arbitrarily small constant ε.

In the case of 3-sided queries, our update time reduces to $O\left(\log^{1/2+\varepsilon} n\right)$, improving Wilkinson's previous bound [ESA 2014] of $O\left(\log^{2/3+\varepsilon} n\right)$.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases dynamic data structures, range searching, computational geometry

Digital Object Identifier 10.4230/LIPIcs.SoCG.2017.28

1 Introduction

Orthogonal range searching is one of the most well-studied and fundamental problems in computational geometry: the goal is to design a data structure to store a set of n points so that we can quickly report all points inside a query axis-aligned rectangle. In the "emptiness" version of the problem, we just want to decide if the rectangle contains any point. (We will not study the counting version of the problem here.)

The static 2-d problem has been extensively investigated [15, 6, 25, 13, 11, 22, 1, 21], with the current best results in the word RAM model given by Chan, Larsen, and Pătraşcu [9].

In this paper, we are interested in the *dynamic* 2-d problem, allowing insertions and deletions of points. A straightforward dynamization of the standard *range tree* [27] supports queries in $O\left(\log^2 n + k\right)$ time and updates in $O\left(\log^2 n\right)$ time, where k denotes the number of reported points (for the emptiness problem, we can take $k = 0$). Mehlhorn and Näher [17] improved the query time to $O\left(\log n \log\log n + k\right)$ and the update time to $O\left(\log n \log\log n\right)$ by *dynamic fractional cascading*.

* This work was done while the first author was at the University of Waterloo, and was partially supported by an NSERC Discovery Grant.

The first data structure to achieve logarithmic query and update (amortized) time was presented by Mortensen [19]. In fact, he obtained *sub*logarithmic bounds in the word RAM model: the query time is $O\left(\frac{\log n}{\log\log n}+k\right)$ and the amortized update time is $O\left(\log^{7/8+\varepsilon} n\right)$ where ε denotes an arbitrarily small positive constant.

On the lower bound side, Alstrup et al. [2] showed that any data structure with t_u update time for 2-d range emptiness requires $\Omega\left(\frac{\log n}{\log(t_u \log n)}\right)$ query time in the cell-probe model. Thus, Mortensen's query bound is optimal for any data structure with polylogarithmic update time. However, it is conceivable that the update time could be improved further while keeping the same query time. Indeed, the $O\left(\log^{7/8+\varepsilon} n\right)$ update bound looks too peculiar to be optimal, one would think.

Let us remark how intriguing this type of "fractional-power-of-log" bound is, which showed up only on a few occasions in the literature. For example, Chan and Pătraşcu [10] gave a dynamic data structure for 1-d rank queries (counting number of elements less than a given value) with $O\left(\frac{\log n}{\log\log n}\right)$ query time and $O\left(\log^{1/2+\varepsilon} n\right)$ update time. Chan and Pătraşcu also obtained more $\sqrt{\log n}$-type results for various offline range counting problems. Another example is Wilkinson's recent paper [24]: he studied a special case of 2-d orthogonal range reporting for *2-sided* and *3-sided* rectangles and obtained a solution with $O\left(\frac{\log n}{\log\log n}+k\right)$ amortized query time, $O\left(\log^{1/2+\varepsilon} n\right)$ update time for the 2-sided case, and $O\left(\log^{2/3+\varepsilon} n\right)$ update time for 3-sided; the latter improves Mortensen's $O\left(\log^{5/6+\varepsilon} n\right)$ update bound for 3-sided [19]. He also showed that in the insertion-only and deletion-only settings, it is possible to get fractional-power-of-log bounds for both the update and the query time. However, he was unable to make progress for general 4-sided rectangles in the insertion-only and deletion-only settings, let alone the fully dynamic setting.

New results. Our main new result is a fully dynamic data structure for 2-d orthogonal range reporting with $O\left(\frac{\log n}{\log\log n}+k\right)$ optimal query time and $O\left(\log^{2/3+o(1)} n\right)$ update time, greatly improving Mortensen's $O\left(\log^{7/8+\varepsilon} n\right)$ bound. In the 3-sided case, we obtain $O\left(\log^{1/2+\varepsilon} n\right)$ update time, improving Wilkinson's $O\left(\log^{2/3+\varepsilon} n\right)$ bound. (See Table 1 for comparison.) Our update bounds seem to reach a natural limit with this type of approach. In particular, it is not unreasonable to conjecture that the near-$\sqrt{\log n}$ update bound for the 3-sided case is close to optimal, considering prior "fractional-power-of-log" upper-bound results in the literature (although there have been no known lower bounds of this type so far).

Like previous methods, our bounds are amortized (this includes query time). Our results are in the word-RAM model, under the standard assumption that the word size w is at least $\log n$ bits (in fact, except for an initial predecessor search during each query/update, we only need operations on $(\log n)$-bit words). Even to researchers uncomfortable with sublogarithmic algorithms on the word RAM, such techniques are still relevant. For example, Mortensen extended his data structure to $d \geq 3$ dimensions and obtained $O\left(\left(\frac{\log n}{\log\log n}\right)^{d-1}+k\right)$ query time and $O\left(\log^{d-9/8+\varepsilon} n\right)$ update time, even in the real-RAM model (where each word can hold an input real number or a $(\log n)$-bit number). Our result automatically leads to improvements in higher dimensions as well.

Table 1 Dynamic planar orthogonal range reporting: previous and new results.

		Update time	Query time
4-sided	Lueker and Willard [27]	$\log^2 n$	$\log^2 n + k$
	Mehlhorn and Näher [17]	$\log n \log\log n$	$\log n \log\log n + k$
	Mortensen [19]	$\log^{7/8+\varepsilon} n$	$\frac{\log n}{\log\log n} + k$
	New	$\log^{2/3} n \log^{O(1)} \log n$	$\frac{\log n}{\log\log n} + k$
3-sided	McCreight [16]	$\log n$	$\log n + k$
	Willard [26]	$\frac{\log n}{\log\log n}$	$\frac{\log n}{\log\log n} + k$
	Mortensen [19]	$\log^{5/6+\varepsilon} n$	$\frac{\log n}{\log\log n} + k$
	Wilkinson [24]	$(\log n \log\log n)^{2/3}$	$\log n + k$
	Wilkinson [24]	$\log^{2/3+\varepsilon} n$	$\frac{\log n}{\log\log n} + k$
	New	$\log^{1/2+\varepsilon} n$	$\frac{\log n}{\log\log n} + k$

Overview of techniques: Micro- and macro-structures. Our solution builds on ideas from Mortensen's paper [19]. His paper was long and not easy to follow, unfortunately; we strive for a clearer organization and a more accessible exposition (which in itself would be a valuable contribution).

The general strategy towards obtaining fractional-power-of-log bounds, in our view, can be broken into two parts: the design of what we will call *micro-structures* and *macro-structures*.

- Micro-structures refer to data structures for handling a small number s of points; by "small", we mean $s = 2^{\log^\alpha n}$ for some fraction $\alpha < 1$ (rather than s being polylogarithmic, as is more usual in other contexts). When s is small, by *rank space reduction* we can make the universe size small, and as a consequence pack multiple points (about $\frac{w}{\log s}$) into a single word. As observed by Chan and Pătraşcu [10] and Wilkinson [24], we can design micro-structures by thinking of each word as a block of multiple points, and borrowing known techniques from the world of *external-memory* algorithms (specifically, *buffer trees* [4]) to achieve *(sub)constant* amortized update time. Alternatively, Mortensen described his micro-structures from scratch, which required a more complicated solution to a certain "pebble game" [19, Section 6].

  One subtle issue is that to simulate rank space reduction dynamically, we need *list labeling* techniques, which, if not carefully implemented, can worsen the exponent in the update bound (as was the case in both Mortensen's and Wilkinson's solutions).
- Macro-structures refer to data structures for large input size n, constructed using micro-structures as black boxes. This part does not involve bit packing, and relies on more traditional geometric divide-and-conquer techniques such as higher-degree range trees, as in Mortensen's and Chan and Pătraşcu's solutions, with degree $2^{\log^\beta n}$ for some fraction $\beta < 1$. Van Emde Boas recursion is also a crucial ingredient in Mortensen's macro-structures.

Our solution will require a number of new ideas in both micro- and macro-structures. On the micro level, we bypass the "pebbling" problem by explicitly invoking external-memory techniques, as in Wilkinson's work [24], but we handle the list labeling issue more carefully, to avoid worsening the update time. On the macro level, we use higher-degree range trees but with a more intricate analysis (involving Harmonic series, interestingly), plus a few bootstrapping steps, in order to achieve the best update and query bounds.

2 Preliminaries

In all our algorithms, we assume that during each query or update, we are given a pointer to the predecessor/successor of the x- and y-values of the given point or rectangle. At the end, we can add the cost of predecessor search to the query and update time (which is no bigger than $O\left(\sqrt{\log n}\right)$ [3] in the word RAM model).

We assume a word RAM model that allows for a constant number of "exotic" operations on w-bit words. By setting $w := \delta \log n$ for a sufficiently small constant δ, these operations can be simulated in constant time by table lookup, after preprocessing the tables in $2^{O(w)} = n^{O(\delta)}$ time.

For simplicity, we concentrate on emptiness queries; all our algorithms can be easily modified for reporting queries, with an additional $O(k)$ term to the query time bounds.

A *3-sided* query deals with a rectangle that is unbounded on the left or right side, by default. A *2-sided* (or *dominance*) query deals with a rectangle that is unbounded on two adjacent sides.

Let $[n]$ denote $\{0, 1, \ldots, n-1\}$.

We now quickly review a few useful tools.

List labeling. *Monotone list labeling* is the problem of assigning *labels* to a dynamic set of totally ordered elements, such that whenever $x < y$, the label of x is less than the label of y. As elements are inserted, we are allowed to change labels. The following result is well known:

▶ **Lemma 1** ([12]). *A monotone labeling for n totally ordered elements with labels in $\left[n^{O(1)}\right]$ can be maintained under insertions by making $O(n \log n)$ label changes.*

Weight-balancing. *Weight-balanced B-trees* [5] are B-tree implementations with a rebalancing scheme that is based on the nodes' *weights*, i.e., subtree sizes, in order to support updates of secondary structures efficiently.

▶ **Lemma 2** ([5], Lemma 4). *In a weight-balanced B-tree of degree s, nodes at height i have weight $\Theta\left(s^i\right)$, and any sequence of n insertions requires at most $O\left(n/s^i\right)$ splits of nodes at height i.*

Colored predecessors. *Colored predecessor searching* is the problem of maintaining a dynamic set of multi-colored, totally ordered elements and searching for the predecessors with a given color.

▶ **Lemma 3** ([19], Theorem 14). *Colored predecessor searches and updates on n colored, totally ordered elements can be supported in $O\left(\log^2 \log n\right)$ time deterministically.*

Van Emde Boas transformation. A crucial ingredient we will use is a general technique of Mortensen [18, 19] that transforms any given data structure for orthogonal range emptiness on small sets of s points, to one for point sets in a *narrow grid* $[s] \times \mathbb{R}$, at the expense of a $\log \log n$ factor increase in cost. We state the result in a slightly more general form:

▶ **Lemma 4** ([19], Theorem 1). *Let X be a set of $O(s)$ values. Given a dynamic data structure for j-sided orthogonal range emptiness ($j \in \{3, 4\}$) on s points in $X \times \mathbb{R}$ with update time $U_j(s)$ and query time $Q_j(s)$, there exists a dynamic data structure for j-sided orthogonal range emptiness on n points in $X \times \mathbb{R}$ with update time $O(U_j(s) \log \log n)$ and query time $O(Q_j(s) \log \log n)$.*

If the given data structure supports updates to X in $U_X(s)$ time and this update procedure depends solely on X (and not the point set), the new data structure can support updates to X in $U_X(s)$ time.

Mortensen's transformation is obtained via a van-Emde-Boas-like recursion [23]: Roughly, we divide the plane into $\sqrt{n}$ horizontal slabs each with $\sqrt{n}$ points; for each slab, we store the topmost and bottommost point at each x-coordinate of X in a data structure for $O(s)$ points, and handle the remaining points recursively. (Note that all these data structures for $O(s)$ points work with a common set X of x-coordinates.)

3 Part 1: Micro-Structures

We first design *micro-structures* for 3- and 4-sided dynamic orthogonal range emptiness when the number of points s is small. This part heavily relies on bit-packing techniques.

3.1 Static universe

We begin with the case of a static universe $\left[s^{O(1)}\right]^2$.

▶ **Lemma 5.** *For s points in the static universe $\left[s^{O(1)}\right]^2$, there exist data structures for dynamic orthogonal range emptiness that support*

(i) *updates in $O\left(\frac{\log^2 s}{w}+1\right)$ amortized time and 3-sided queries in $O(\log s)$ amortized time;*

(ii) *updates in $O\left(\frac{\log^3 s}{w}+1\right)$ amortized time and 4-sided queries in $O\left(\log^2 s\right)$ amortized time.*

Proof. We mimick existing *external-memory* data structures with a block size of $B := \left\lceil \frac{\delta w}{\log s} \right\rceil$ for a sufficiently small constant δ, observing that B points can be packed into a single word.

(i) For the 3-sided case, Wilkinson [24, Lemma 1] has already adapted such an external-memory data structure, namely, a *buffered* version of a binary *priority search tree* due to Kumar and Schwabe [14] (see also Brodal's more recent work [7]), which is similar to the *buffer tree* of Arge [4]. For 3-sided rectangles unbounded to the left/right, the priority search tree is ordered by y, where each node stores $O(B)$ x-values. Wilkinson obtained $O\left(\frac{1}{B}\cdot\log s+1\right) = O\left(\frac{\log^2 s}{w}+1\right)$ amortized update time and $O(\log s)$ amortized query time.

(ii) For the general 4-sided case, we use a *buffered* version of a binary *range tree*. Although we are not aware of prior work explicitly giving such a variant of the range tree, the modifications are straightforward, and we will provide only a rough outline. The range tree is ordered by y. Each node holds a buffer of up to B update requests that have not yet been processed. Each node is also augmented with a 1-d binary *buffer tree* (already described by Arge [4]) for the x-projection of the points. To insert or delete a point, we add the update request to the root's buffer. Whenever a buffer's size of a node exceeds B, we empty the buffer by applying the following procedure: we divide the list of $\Theta(B)$ update requests into two sublists for the two children in $O(1)$ time using an exotic word operation (since B update requests fit in a word); we then pass these sublists to the buffers at the two children, and also pass another copy of the list to the node's 1-d buffer tree. These 1-d updates cost $O\left(\frac{1}{B}\cdot\log s\right)$ each [4], when amortized over $\Omega(B)$ updates. Since each update eventually travels to $O(\log s)$ nodes of the range tree, the amortized update time of the 4-sided structure is $O\left(\frac{1}{B}\log^2 s+1\right) = O\left(\frac{\log^3 s}{w}+1\right)$.

A 4-sided query is answered by following two paths in the range tree in a top-down manner, performing $O(\log s)$ 1-d queries; since each 1-d query takes $O(\log s)$ time, the

overall query time is $O\left(\log^2 s\right)$. However, before we can answer the query, we need to first empty the buffers along the two paths of the range tree. This can be done by applying the procedure in the preceding paragraph at the $O(\log s)$ nodes top-down; this takes $O(\log s)$ time, plus the time needed for $O(B \log s)$ 1-d updates, costing $O\left(\frac{1}{B} \cdot \log s\right)$ each [4]. The final amortized query time is thus $O\left(\log^2 s\right)$. ◀

Notice that the above update time is *constant* when the number of points s is as large as $2^{\sqrt{w}}$ for 3-sided queries or $2^{w^{1/3}}$ for 4-sided.

(It is possible to eliminate one of the logarithmic factors in the query time for the above 4-sided result, by augmenting nodes of the range tree with 3-sided structures. However, this alternative causes difficulty later in the extension to dynamic universes. Besides, the larger query time turns out not to matter for our macro-structures at the end.)

3.2 Dynamic universe

To make the preceding data structure support a dynamic universe, the simplest way is to apply monotone list labeling (Lemma 1), which maps coordinates to $\left[s^{O(1)}\right]^2$. Whenever a label of a point changes, we just delete the point and reinsert a copy with the new coordinates into the data structure. However, since the total number of label changes is $O(s \log s)$ over s insertions, this slows down the amortized update time by a $\log s$ factor and will hurt the final update bound.

Our approach is as follows. We first observe that the list labeling approach works fine for changes to the y-universe. For changes to the x-universe, we switch to a "brute-force" method with large running time, but luckily, since the number of such changes will be relatively small, this turns out to be adequate for our macro-structures at the end. (The brute-force idea can also be found in Mortensen's paper [19], but his macro-structures were less efficient.)

▶ **Lemma 6.** *Both data structures in Lemma 5 can be modified to work for s points in a universe $X \times Y$ with $|X|, |Y| = O(s)$. The update and query time bounds are the same, and we can support*

- **(i)** *updates to Y in $O\left(\log^2 \log s\right)$ amortized time (given a pointer to the predecessor/successor in Y), and*
- **(ii)** *updates to X in $2^{O(w)}$ time, where the update procedure for X depends solely on X (and not the point set).*

Proof. (i) To start, let us assume that $X = \left[s^{O(1)}\right]$ but Y is arbitrary. We divide the sorted list Y into $O(s/A)$ blocks of size $\Theta(A)$ for a parameter A to be set later. It is easy to maintain such a blocking using $O(s/A)$ number of block merges and splits over s updates. (Such a blocking was also used by Wilkinson [24].) We maintain a monotone labeling of the blocks by Lemma 1. In the proof of Lemma 5, we construct the y-ordered priority search tree or range tree using the block labels as the y-values. Each leaf then corresponds to a block. We build a small range tree for each leaf block to support updates and queries for the $O(A)$ points in, say, $O\left(\log^2 A\right)$ time. We can encode a y-value $\eta \in Y$ by a pair consisting of the label of the block containing η, and the rank of η with respect to the block. We will use these encoded values, which still are $O(\log s)$-bit long, in all the buffers. The block labels provide sufficient information to pass the update requests to the leaves and the x-ordered 1-d buffer trees. The ranks inside a block provide sufficient information to handle a query or update at a leaf.

During each block split/merge and each block label change, we need to first empty the buffers along the path to the block before applying the change. This can be done by applying

the procedure from the proof of Lemma 5 at $O(\log s)$ nodes top-down, requiring $O(\log s)$ amortized time. Since the total number of block label changes is $O\left(\frac{s}{A}\log\frac{s}{A}\right)$, the total time for these steps is $O\left(\frac{s}{A}\log\frac{s}{A}\cdot\log s\right) = O(s)$ by setting $A := \log^2 s$. The amortized cost for these steps is thus $O(1)$.

(ii) Now, we remove the $X = \left[s^{O(1)}\right]$ assumption. We assign elements in X to labels in $[O(s)]$ but do not insist on a monotone labeling. Then no label change is necessary! We will use these labels for the x-values in all the buffers. The exotic word operations are simulated by table lookup, but in the precomputation of each table entry, we need to first map the labels to their actual x-values. During each update to X, we now need to recompute all table entries by brute force, taking $2^{O(w)}$ time. ◀

4 Part 2: Macro-Structures

We now present macro-structures for 3- and 4-sided dynamic orthogonal range emptiness when the number of points n is large, by using micro-structures as black boxes. This part does not involve bit packing (and hence is more friendly to computational geometers). The transformation from micro- to macro-structures is based on variants of range trees.

4.1 Range tree transformation I

▶ **Lemma 7.** *Given a family of data structures $\mathcal{D}_j^{(i)}$ $(i \in \{1,\ldots,\log_s n\})$ for dynamic j-sided orthogonal range emptiness $(j \in \{3,4\})$ on s points in $X \times \mathbb{R}$ $(|X| = O(s))$ with update time $U_j^{(i)}(s)$ and query time $Q_j^{(i)}(s)$, where updates to X take $U_X^{(i)}(s)$ time with a procedure that depends solely on X, there exist data structures for dynamic orthogonal range emptiness on n points in the plane with the following amortized update and query time:*

(i) *for the 3-sided case,*

$$U_3'(n) = O\left(\sum_{i=1}^{\log_s n} U_3^{(i)}(s)\log\log n + \sum_{i=1}^{\log_s n}\frac{U_X^{(i)}(s)}{s^{i-1}} + \log_s n\log^2\log n\right)$$

$$Q_3'(n) = O\left(\max_i Q_3^{(i)}(s)\log_s n\log\log n + \log_s n\log^2\log n\right);$$

(ii) *for the 4-sided case,*

$$U_4'(n) = O\left(\sum_{i=1}^{\log_s n}(U_4^{(i)}(s)+U_3^{(i)}(s))\log\log n + \sum_{i=1}^{\log_s n}\frac{U_X^{(i)}(s)}{s^{i-1}} + \log_s n\log^2\log n\right)$$

$$Q_4'(n) = O\left(\max_i Q_4^{(i)}(s)\log\log n + \max_i Q_3^{(i)}(s)\log_s n\log\log n + \log_s n\log^2\log n\right).$$

Proof. We store a range tree ordered by x, implemented as a degree-s weight-balanced B-tree. (Deletions can be handled lazily without changing the weight-balanced tree; we can rebuild periodically when n decreases or increases by a constant factor.) At every internal node v at height i, we store the points in its subtree in a data structure for j-sided orthogonal range emptiness on a *narrow grid* $X_v \times \mathbb{R}$, obtained by applying Lemma 4 to the given structure $\mathcal{D}_j^{(i)}$, where X_v is the set of x-coordinates of the $O(s)$ dividing vertical lines at the node, and the x-coordinate of every point is replaced with the predecessor in X_v. We also store the y-coordinates of these points in a colored predecessor searching structure of Lemma 3, where points in the same child's vertical slab are assigned the same color. And we store the x-coordinates in another colored predecessor searching structure, where X_v is colored black and the rest is colored white.

To insert or delete a point, we update the narrow-grid structures at the nodes along the path in the tree. This takes $O\left(\sum_{i=1}^{\log_s n} U_j^{(i)}(s) \log\log n\right)$ total time. Note that given the y-predecessor/successor of the point at a node, we can obtain the y-predecessor/successor at the child by using the colored predecessor searching structure. We can also determine the x-predecessor in X_v by another colored predecessor search. This takes total time $O\left(\log_s n \log^2 \log n\right)$ along the path.

To keep the tree balanced, we need to handle node splits. For nodes at height i, there are $O\left(n/s^i\right)$ splits by Lemma 2. Each such split requires rebuilding two narrow-grid structures on $O\left(s^i\right)$ points, which can be done naively by $O\left(s^i\right)$ insertions to empty structures. This has $O\left(\sum_{i=1}^{\log_s n} \left(n/s^i\right) \cdot s^i U_j^{(i)}(s) \log\log n\right)$ total cost, i.e., an amortized cost of $O\left(\sum_{i=1}^{\log_s n} U_j^{(i)}(s) \log\log n\right)$. A split of a child of v also requires updating (deleting and reinserting) the points at the child's slab. This has $O\left(\sum_{i=1}^{\log_s n} \left(n/s^{i-1}\right) \cdot s^{i-1} U_j^{(i)}(s) \log\log n\right)$ total cost, i.e., an amortized cost of $O\left(\sum_{i=1}^{\log_s n} U_j^{(i)}(s) \log\log n\right)$. Furthermore, a split of a child of v requires an update to X_v. This has $O\left(\sum_{i=1}^{\log_s n} \left(n/s^{i-1}\right) \cdot U_X^{(i)}(s)\right)$ total cost, i.e., an amortized cost of $O\left(\sum_{i=1}^{\log_s n} \left(1/s^{i-1}\right) \cdot U_X^{(i)}(s)\right)$.

To answer a 3-sided query, we proceed down a path of the tree and perform queries in the narrow-grid structures at nodes along the path. This takes $O\left(\log_s n \cdot \max_i Q_3^{(i)}(s) \log\log n\right)$ total time. As before, given the y-predecessor/successor of the coordinates of the rectangle at a node, we can obtain the y-predecessor/successor at the child by using the colored predecessor searching structure. This takes total time $O\left(\log_s n \log^2 \log n\right)$ along the path.

To answer a 4-sided query, we find the highest node v whose dividing vertical lines cut the query rectangle. We obtain two 3-sided queries at two children of v, which can be answered as above, plus a remaining query that can be answered via the narrow-grid structure at v in $O\left(\max_i Q_4^{(i)}(s) \log\log n\right)$ time. ◀

Combining with our preceding micro-structures, we obtain the following results, achieving the desired update time but slightly suboptimal query time (which we will fix later):

▶ **Theorem 8.** *Given n points in the plane, there exist data structures for dynamic orthogonal range emptiness that support*

- **(i)** *updates in amortized $O\left(\log^{1/2} n \log^{O(1)} \log n\right)$ time and 3-sided queries in amortized $O\left(\log n \log\log n\right)$ time;*
- **(ii)** *updates in amortized $O\left(\log^{2/3} n \log^{O(1)} \log n\right)$ time and 4-sided queries in amortized $O\left(\log n \log\log n\right)$ time.*

Proof. (i) For the 3-sided case, Lemmata 5(i) and 6 give micro-structures with update time $O\left(\frac{\log^2 s}{\overline{w}} + \log^2 \log s\right)$ and query time $O\left(\log s\right)$, while supporting updates to X in $2^{O(\overline{w})}$ time. Observe that we can choose to work with a smaller word size $\overline{w} \leq w$, so long as $\overline{w} = \Omega\left(\log s\right)$. We choose $\overline{w} := \delta i \log s$ for a sufficiently small absolute constant δ and for any given $i \in [2, \log_s n]$. This gives

$$\begin{aligned} U_3^{(i)}(s) &= O\left(\frac{\log s}{i} + \log^2 \log s\right) \\ Q_3^{(i)}(s) &= O\left(\log s\right) \\ U_X^{(i)}(s) &= s^{O(\delta i)}. \end{aligned}$$

For the special case $i = 1$, we use a standard priority search tree, achieving $U_3^{(1)}(s), Q_3^{(1)}(s) = O(\log s)$ and $U_X^{(1)}(s) = 0$. Substituting into Lemma 7, we obtain

$$\begin{aligned} U_3'(n) &= O\left(\sum_{i=1}^{\log_s n} \frac{\log s \log\log n}{i} + \log_s n \log^3 \log n + \sum_{i=2}^{\log_s n} \frac{s^{O(\delta i)}}{s^{i-1}} \log_s n \log^2 \log n\right) \\ &= O\left(\log s \log^2 \log n + \log_s n \log^3 \log n\right), \end{aligned}$$

since the first sum is a Harmonic series and the second sum is a geometric series. (This assumes a sufficiently small constant for δ, as the hidden constant in the exponent $O(\delta i)$ does not depend on δ.) Furthermore,

$$\begin{aligned} Q_3'(n) &= O\left(\log s \log_s n \log\log n + \log_s n \log^2 \log n\right) \\ &= O\left(\log n \log\log n + \log_s n \log^2 \log n\right). \end{aligned}$$

We set $s := 2^{\sqrt{\log n}}$ to get $U_3'(n) = O\left(\log^{1/2} n \log^{O(1)} \log n\right), Q_3'(n) = O(\log n \log\log n)$.

(ii) Similarly, for the 4-sided case, Lemmata 5(ii) and 6 with a smaller word size $\overline{w} := \delta i \log s$ give micro-structures with

$$\begin{aligned} U_4^{(i)}(s) &= O\left(\frac{\log^2 s}{i} + \log^2 \log s\right) \\ Q_4^{(i)}(s) &= O\left(\log^2 s\right) \\ U_X^{(i)}(s) &= s^{O(\delta i)}. \end{aligned}$$

For the special case $i = 1$, we use a standard range tree, achieving $U_4^{(1)}(s), Q_4^{(1)}(s) = O\left(\log^2 s\right)$ and $U_X^{(1)}(s) = 0$. Substituting into Lemma 7, we obtain

$$\begin{aligned} U_4'(n) &= O\left(\sum_{i=1}^{\log_s n} \frac{\log^2 s \log\log n}{i} + \log_s n \log^3 \log n + \sum_{i=2}^{\log_s n} \frac{s^{O(\delta i)}}{s^{i-1}} \log_s n \log^2 \log n\right) \\ &= O\left(\log^2 s \log^2 \log n + \log_s n \log^3 \log n\right) \end{aligned}$$

and

$$\begin{aligned} Q_4'(n) &= O\left(\log^2 s \log\log n + \log s \log_s n \log\log n + \log_s n \log^2 \log n\right) \\ &= O\left(\log^2 s \log\log n + \log n \log\log n + \log_s n \log^2 \log n\right). \end{aligned}$$

We set $s := 2^{\log^{\frac{1}{3}} n}$ to get $U_4'(n) = O\left(\log^{2/3} n \log^{O(1)} \log n\right), Q_4'(n) = O(\log n \log\log n)$. ◀

4.2 Range tree transformation II

We now reduce the query time to optimal by another transformation:

▶ **Lemma 9.** *Given data structures for dynamic j-sided orthogonal range emptiness ($j \in \{2, 3, 4\}$) on n points in the plane with update time $U_j(n)$ and query time $Q_j(n)$, there exist data structures for dynamic j-sided orthogonal range emptiness ($j \in \{3, 4\}$) on n points in the plane with the following amortized update and query time:*

$$\begin{aligned} U_j'(n) &= O\left(U_j(s) \log_s n \log\log n + U_{j-1}(n) \log_s n + \log_s n \log^2 \log n\right) \\ Q_j'(n) &= O\left(Q_j(s) \log\log n + Q_{j-1}(n) + \log_s n \log^2 \log n\right). \end{aligned}$$

Proof. We first switch the x- and y-coordinates of the points. This is fine since the given data structures in the statement of this lemma still exist by symmetry (unlike in Lemma 7).

We modify the range tree in the proof of Lemma 7, where every internal node is augmented with a $(j-1)$-sided structure.

During an insertion or deletion of a point, we update the narrow-grid structures along a path as before, in $O(\log_s n \cdot U_j(s) \log\log n)$ time. We now also need to update the $(j-1)$-sided structures at nodes along the path. This adds $O(U_{j-1}(n) \log_s n)$ to the update time.

During rebalancing, each split of a node at height i now requires rebuilding the $(j-1)$-sided structures, which can be done naively by $O(s^i)$ insertions to an empty structure. This has $O\left(\sum_{i=1}^{\log_s n} (n/s^i) \cdot s^i U_{j-1}(n)\right)$ total cost, i.e., an amortized cost of $O(U_{j-1}(n) \log_s n)$.

To answer a j-sided query, we find the highest node v whose dividing vertical lines cut the query rectangle. We obtain two $(j-1)$-sided queries at two children of v, plus a query in the narrow-grid structure at v. (In the case $j = 3$, recall that the input to a 3-sided query is now a rectangle unbounded from above or below, because of the switching of x and y.) The two $(j-1)$-sided queries can be answered directly using the augmented structures. This takes $O(Q_j(s) \log\log n + Q_{j-1}(n))$ time, plus the cost $O(\log_s n \log^2 \log n)$ to descend along the path to that node. ◀

We obtain our final results by bootstrapping:

▶ **Theorem 10.** *Given n points in the plane, there exist data structures for dynamic orthogonal range emptiness that support*

- **(i)** *updates in amortized $O\left(\log^{1/2+O(\varepsilon)} n\right)$ time and 3-sided queries in amortized $O\left(\frac{\log n}{\log\log n}\right)$ time;*
- **(ii)** *updates in amortized $O\left(\log^{2/3} n \log^{O(1)} \log n\right)$ time and 4-sided queries in amortized $O\left(\frac{\log n}{\log\log n}\right)$ time.*

Proof. (i) Theorem 8(i) achieves

$$\begin{aligned} U_3(s) &= O\left(\log^{1/2} s \log^{O(1)} \log s\right) \\ Q_3(s) &= O(\log s \log\log s). \end{aligned}$$

Wilkinson [24] has given a data structure for 2-sided (dominance) queries with

$$\begin{aligned} U_2(n) &= O\left(\log^{1/2+\varepsilon} n\right) \\ Q_2(n) &= O\left(\frac{\log n}{\log\log n}\right). \end{aligned}$$

Substituting into Lemma 9, we obtain

$$\begin{aligned} U_3'(n) &= O\left(\log^{1/2} s \log_s n \log^{O(1)} \log n \;+\; \log^{1/2+\varepsilon} n \log_s n + \log_s n \log^2 \log n\right) \\ Q_3'(n) &= O\left(\log s \log\log s \log\log n \;+\; \frac{\log n}{\log\log n} \;+\; \log_s n \log^2 \log n\right). \end{aligned}$$

We set $s := 2^{\frac{\log n}{\log^3 \log n}}$ to get $U_3'(n) = O\left(\log^{1/2+O(\varepsilon)} n\right), Q_3'(n) = O\left(\frac{\log n}{\log\log n}\right)$.

(ii) Similarly, Theorem 8(ii) achieves

$$\begin{aligned} U_4(s) &= O\left(\log^{2/3} s \log^{O(1)} \log s\right) \\ Q_4(s) &= O(\log s \log\log s). \end{aligned}$$

Part (i) above gives

$$\begin{aligned} U_3(n) &= O\left(\log^{1/2+O(\varepsilon)} n\right) \\ Q_3(n) &= O\left(\frac{\log n}{\log\log n}\right). \end{aligned}$$

Substituting into Lemma 9, we obtain

$$\begin{aligned} U'_4(n) &= O\left(\log^{2/3} s \log_s n \log^{O(1)} \log n \;+\; \log^{1/2+O(\varepsilon)} n \log_s n + \log_s n \log^2 \log n\right) \\ Q'_4(n) &= O\left(\log s \log\log s \log\log n \;+\; \frac{\log n}{\log\log n} \;+\; \log_s n \log^2 \log n\right). \end{aligned}$$

We set $s := 2^{\frac{\log n}{\log^3 \log n}}$ to get $U'_4(n) = O\left(\log^{2/3} n \log^{O(1)} \log n\right), Q'_4(n) = O\left(\frac{\log n}{\log\log n}\right)$. ◀

5 Future Work

We have not yet mentioned space complexity. We can trivially upper-bound the space of our data structure by n times the update time, i.e., $O\left(n \log^{2/3+o(1)} n\right)$ for the 4-sided case, which is already an improvement over Mortensen's $O\left(n \log^{7/8+\varepsilon} n\right)$ space bound. We are currently working on ways to improve space further to near-linear. (See [20, 21] for the current best data structures with near-linear space.)

We can automatically extend our result to higher constant dimensions $d \geq 3$ by using a standard degree-b range tree, which adds a $b \log_b n$ factor per dimension to the update time and a $\log_b n$ factor per dimension to the query time. With $b = \log^{\varepsilon} n$, this gives $O\left((\log n/\log\log n)^{d-1}\right)$ query time and $O\left(\log^{d-5/3+O(\varepsilon)} n\right)$ update time, improving Mortensen's result. Alternatively, we can directly modify our micro- and macro-structures, which should give a better update time of the form $O\left(\log^{d-2+O(1/d)} n\right)$. We are currently working on obtaining the best precise exponent with this approach. (See [8] for a different tradeoff with query time better by about a logarithmic factor but update time worse by several logarithmic factors.)

References

1. Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. New data structures for orthogonal range searching. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 198–207, 2000. doi:10.1109/SFCS.2000.892088.
2. Stephen Alstrup, Thore Husfeldt, and Theis Rauhe. Marked ancestor problems. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 534–543, Nov 1998. doi:10.1109/SFCS.1998.743504.
3. Arne Andersson and Mikkel Thorup. Dynamic ordered sets with exponential search trees. *Journal of the ACM*, 54(3):13, 2007. doi:10.1145/1236457.1236460.
4. Lars Arge. The buffer tree: A technique for designing batched external data structures. *Algorithmica*, 37(1):1–24, 2003. doi:10.1007/s00453-003-1021-x.
5. Lars Arge and Jeffrey Scott Vitter. Optimal external memory interval management. *SIAM Journal on Computing*, 32(6):1488–1508, 2003. doi:10.1137/S009753970240481X.
6. Jon Louis Bentley. Decomposable searching problems. *Information Processing Letters*, 8(5):244–251, 1979. doi:10.1016/0020-0190(79)90117-0.

7 Gerth Stølting Brodal. External memory three-sided range reporting and top-k queries with sublogarithmic updates. In *Proceedings of the 33rd Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 23:1–23:14, 2016. doi:10.4230/LIPIcs.STACS.2016.23.

8 Timothy M. Chan. Three problems about dynamic convex hulls. *International Journal of Computational Geometry and Applications*, 22(4):341–364, 2012. doi:10.1142/S0218195912600096.

9 Timothy M. Chan, Kasper Green Larsen, and Mihai Pătraşcu. Orthogonal range searching on the RAM, revisited. In *Proceedings of the 27th Annual Symposium on Computational Geometry (SoCG)*, pages 1–10, 2011. doi:10.1145/1998196.1998198.

10 Timothy M. Chan and Mihai Pătraşcu. Counting inversions, offline orthogonal range counting, and related problems. In *Proceedings of the 21st Annual ACM–SIAM Symposium on Discrete Algorithms (SODA)*, pages 161–173, 2010. doi:10.1137/1.9781611973075.15.

11 Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal on Computing*, 17(3):427–462, 1988. doi:10.1137/0217026.

12 Paul F. Dietz. Maintaining order in a linked list. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing (STOC)*, pages 122–127, 1982. doi:10.1145/800070.802184.

13 Otfied Fries, Kurt Mehlhorn, Stefan Näher, and Athanasios K. Tsakalidis. A $\log \log n$ data structure for three-sided range queries. *Information Processing Letters*, 25(4):269–273, 1987. doi:10.1016/0020-0190(87)90174-8.

14 Vijay Kumar and Eric J. Schwabe. Improved algorithms and data structures for solving graph problems in external memory. In *Proceedings of the 8th Annual IEEE Symposium on Parallel and Distributed Processing*, pages 169–176, 1996. doi:10.1109/SPDP.1996.570330.

15 George S. Lueker. A data structure for orthogonal range queries. In *Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 28–34, 1978. doi:10.1109/SFCS.1978.1.

16 Edward M. McCreight. Priority search trees. *SIAM Journal on Computing*, 14(2):257–276, 1985. doi:10.1137/0214021.

17 Kurt Mehlhorn and Stefan Näher. Dynamic fractional cascading. *Algorithmica*, 5(1):215–241, 1990. doi:10.1007/BF01840386.

18 Christian Worm Mortensen. Fully-dynamic two dimensional orthogonal range and line segment intersection reporting in logarithmic time. In *Proceedings of the 14th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA)*, pages 618–627, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644210".

19 Christian Worm Mortensen. Fully dynamic orthogonal range reporting on RAM. *SIAM Journal on Computing*, 35(6):1494–1525, 2006. doi:10.1137/S0097539703436722.

20 Yakov Nekrich. Space efficient dynamic orthogonal range reporting. *Algorithmica*, 49(2):94–108, 2007. doi:10.1007/s00453-007-9030-9.

21 Yakov Nekrich. Orthogonal range searching in linear and almost-linear space. *Computational Geometry*, 42(4):342–351, 2009. doi:10.1016/j.comgeo.2008.09.001.

22 Mark H. Overmars. Efficient data structures for range searching on a grid. *Journal of Algorithms*, 9(2):254–275, 1988. doi:10.1016/0196-6774(88)90041-7.

23 Peter van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information Processing Letters*, 6(3):80–82, 1977. doi:10.1016/0020-0190(77)90031-X.

24 Bryan T. Wilkinson. Amortized bounds for dynamic orthogonal range reporting. In *Proceedings of the 22th Annual European Symposium on Algorithms (ESA)*, pages 842–856, 2014. doi:10.1007/978-3-662-44777-2_69.

25 Dan E. Willard. New data structures for orthogonal range queries. *SIAM Journal on Computing*, 14(1):232–253, 1985. doi:10.1137/0214019.

26 Dan E. Willard. Examining computational geometry, Van Emde Boas trees, and hashing from the perspective of the fusion tree. *SIAM Journal on Computing*, 29(3):1030–1049, 2000. doi:10.1137/S0097539797322425.

27 Dan E. Willard and George S. Lueker. Adding range restriction capability to dynamic data structures. *Journal of the ACM*, 32(3):597–617, 1985. doi:10.1145/3828.3839.