Dynamic Planar Orthogonal Point Location in Sublogarithmic Time

Timothy M. Chan
Department of Computer Science, University of Illinois at Urbana-Champaign, USA
tmc@illinois.edu

Konstantinos Tsakalidis[1]
Tandon School of Engineering, New York University, USA
kt79@nyu.edu

Abstract

We study a longstanding problem in computational geometry: dynamic 2-d orthogonal point location, i.e., vertical ray shooting among n horizontal line segments. We present a data structure achieving $O\left(\frac{\log n}{\log\log n}\right)$ optimal expected query time and $O\left(\log^{1/2+\varepsilon} n\right)$ update time (amortized) in the word-RAM model for any constant $\varepsilon > 0$, under the assumption that the x-coordinates are integers bounded polynomially in n. This substantially improves previous results of Giyora and Kaplan [SODA 2007] and Blelloch [SODA 2008] with $O(\log n)$ query and update time, and of Nekrich (2010) with $O\left(\frac{\log n}{\log\log n}\right)$ query time and $O\left(\log^{1+\varepsilon} n\right)$ update time. Our result matches the best known upper bound for simpler problems such as dynamic 2-d dominance range searching.

We also obtain similar bounds for orthogonal line segment intersection reporting queries, vertical ray stabbing, and vertical stabbing-max, improving previous bounds, respectively, of Blelloch [SODA 2008] and Mortensen [SODA 2003], of Tao (2014), and of Agarwal, Arge, and Yi [SODA 2005] and Nekrich [ISAAC 2011].

2012 ACM Subject Classification Theory of computation → Computational geometry, Theory of computation → Data structures design and analysis

Keywords and phrases dynamic data structures, point location, word RAM

Digital Object Identifier 10.4230/LIPIcs.SoCG.2018.25

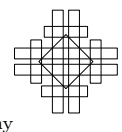
Acknowledgements We would like to thank Athanasios Tsakalidis for helpful discussions.

1 Introduction

Point location is one of the most well-studied and fundamental data structure problems in computational geometry. The static version of the problem dates back to the early years of the field. The *dynamic* version, which supports updates, has also received considerable attention, though obtaining $O(\log n)$ query and update time in 2-d remains open to this day; see [9] for the most recent breakthrough (and the extensive history).

There are two common formulations of 2-d point location. In the first, we want to store a connected planar subdivision with n edges so that we can quickly determine (the label of the) region containing any given query point q; updates correspond to insertions and deletions of edges. In the second formulation, also known as *vertical ray shooting*, we want to store a set of n disjoint line segments so that we can quickly report the lowest segment above

[1] Partially supported by NSF grants CCF-1319648 and CCF-1533564.



any given query point q; updates correspond to insertions and deletions of segments. Since knowing the segment immediately above q allows us to infer the region containing q, the first formulation reduces to the second, at least in the static case.[2] As in many other papers in this area, we focus only on the second formulation, which in some sense is more general since the segments do not need to be connected.

In this paper, we are interested in the *orthogonal* setting of the problem: in the vertical ray shooting formulation, the requirement is that all input line segments are horizontal. This case is among the most basic and important since many applications require handling only orthogonal input. The case when segments have a constant number of different slopes can also be reduced to the orthogonal case, due to the decomposability of vertical ray shooting (we can treat each slope class separately and apply a shear transform to each class).

Classical segment trees can solve the dynamic orthogonal problem with $O(\log^2 n)$ query and update time. In the late 1980s, Mehlhorn and Näher [18] improved the query and update bounds to $O(\log n \log\log n)$ by dynamic fractional cascading. At SODA'07 and SODA'08 respectively, Giyora and Kaplan [16] and Blelloch [6] both obtained $O(\log n)$ query and update time. Earlier at SODA'03, Mortensen [19] obtained $O(\log n)$ query and update time for the *decision* version of vertical ray shooting, namely, *vertical segment intersection emptiness* – deciding whether a vertical query segment intersects any of the horizontal input segments. These results are in the standard $(\log n)$-bit RAM model.

Sublogarithmic time? However, logarithmic time bounds are not the end of the story in the RAM model. For example, for the static orthogonal problem, Chan [8] at SODA'11 presented a linear-space data structure with $O(\log\log N)$ time if both x- and y-coordinates are integers bounded by N.

For the dynamic orthogonal problem, Alstrup et al. [2] applied Fredman and Saks' lower-bound technique [15] to show that any data structure with t_u update time requires $\Omega\left(\frac{\log n}{\log(t_u \log n)}\right)$ query time in the cell-probe model (with $(\log n)$-bit cells). In particular, any data structure with polylogarithmic update time requires $\Omega\left(\frac{\log n}{\log\log n}\right)$ query time. Nekrich [23] has shown that $O\left(\frac{\log n}{\log\log n}\right)$ query time is possible with a data structure for dynamic 2-d orthogonal point location supporting $O\left(\log^{1+\varepsilon} n\right)$ update time. But could we obtain $O\left(\frac{\log n}{\log\log n}\right)$ optimal query time and $O\left(\frac{\log n}{\log\log n}\right)$ update time? Or more ambitiously, could we obtain the same optimal query time with substantially sublogarithmic update time? Alstrup et al.'s lower bound does not rule out this possibility.

Indeed, a phenomenon of "fractional-power-of-log" update times has been observed for several problems with Fredman–Saks-style lower bounds. For example, for dynamic 1-d *rank* queries (or equivalently, 1-d range counting) and *selection* queries, Chan and Pătraşcu [10] obtained a data structure with $O\left(\frac{\log n}{\log\log n}\right)$ optimal query time and $O\left(\log^{1/2+\varepsilon} n\right)$ amortized update time, where $\varepsilon > 0$ denotes an arbitrarily small constant. For dynamic 2-d *orthogonal range searching*, Mortensen in 2006 [21] gave a data structure with $O\left(\frac{\log n}{\log\log n}\right)$ optimal query time and $O\left(\log^{5/6+\varepsilon} n\right)$ amortized update time in the special case of 3-sided queries, or $O\left(\log^{7/8+\varepsilon} n\right)$ amortized update time in general. Wilkinson in 2014 [29] improved

[2] The reduction also holds in the dynamic case, by storing the edges around each region in an "ordered union-split-find" structure [17]. Unfortunately, such a structure requires logarithmic overhead and is inadequate for us, as we aim for sublogarithmic bounds.

Mortensen's update time in the 3-sided case to $O\left(\log^{2/3+\varepsilon} n\right)$, and obtained $O\left(\log^{1/2+\varepsilon} n\right)$ update time in the 2-sided case. Our SoCG paper last year [12] improved these update times further, to $O\left(\log^{1/2+\varepsilon} n\right)$ in the 3-sided case, and $O\left(\log^{2/3+o(1)} n\right)$ in general, while retaining $O\left(\frac{\log n}{\log\log n}\right)$ optimal query time.

Orthogonal range searching and vertical ray shooting are related: 3-sided orthogonal range searching is equivalent to the *1-sided* special case of vertical ray shooting where all input segments are horizontal rays. Mortensen's Ph.D. thesis [20] combined both his papers on range searching [21] and segment intersection emptiness/reporting [19], and it is interesting to note that he was able to obtain fractional-power-of-log update times for the former but not for the latter in general, suggesting that dynamic 2-d orthogonal point location might be a tougher problem than dynamic 2-d orthogonal range searching.

New result. We succeed in simultanesouly obtaining sublogarithmic query time and substantially sublogarithmic update time for dynamic 2-d orthogonal point location (in the vertical ray shooting formulation), under the assumption that x-coordinates are integers bounded polynomially in n. Our data structure achieves $O\left(\frac{\log n}{\log\log n}\right)$ optimal query time and $O\left(\log^{1/2+\varepsilon} n\right)$ update time, greatly improving the previous results of Giyora and Kaplan [16], Blelloch [6], Mortensen [19], and Nekrich [23]. 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). Both our query and update bounds are amortized. Our query time bound is expected: randomization is used, but only in the query algorithm, not in the update algorithm. For the decision problem, vertical segment intersection emptiness, our algorithm is completely deterministic. Our algorithm can be extended to solve vertical segment intersection reporting – reporting all horizontal input segments intersecting a vertical query segment – in $O\left(\frac{\log n}{\log\log n} + k\right)$ deterministic time where k is the number of output segments.

Interestingly, our update time bound is even better than our earlier $O\left(\log^{2/3+o(1)} n\right)$ result [12] for general 2-d orthogonal range searching. There are reasons to believe that $\log^{1/2+\varepsilon} n$ could be the best possible, under current knowledge: The current best result for the simpler problem of 2-d orthogonal *2-sided* (i.e., *dominance*) range searching by Wilkinson [29] already has $O\left(\log^{1/2+\varepsilon} n\right)$ update time; this simpler problem corresponds to the special case of 2-d orthogonal point location where all input line segments and query line segments are rays. Any improvement of our update time would require improvement in this special case first. Besides, sub-$\sqrt{\log n}$ update upper bounds have never been obtained before for *any* problem with Fredman–Saks-style lower bounds.

The assumption of a polynomially bounded x-universe is reasonable and holds in most applications. For example, in offline settings where we know the coordinates of all the input segments in advance, we can simply replace coordinates by their ranks. The known lower bounds still hold in the bounded universe setting. The assumption arises from a technical issue in ensuring balance in our underlying tree structure. It is plausible that it could be eliminated with more technical effort, but we would rather prefer keeping the solution simpler.

Overview of techniques. Following our earlier approach [12] for dynamic orthogonal range searching (which in turn was a simplification of the approach of Mortensen [21]), our general strategy consists of three parts:

1. We first design efficient *micro-structures* for solving the problem for small input of size s, with the goal of achieving near-constant amortized update time. Following [12, 29], this part is obtained by taking an external-memory version of the segment tree, and re-implementing it in internal memory using bit-packing tricks. We use ideas from *buffer trees* [4] to aim for an amortized update cost of the form $O((\log s)/B)$, where B is the block size. Since we can pack $B \approx (\log s)/w$ segments in one word in internal memory, this would yield update time $O((\log^2 s)/w)$, which is indeed small when $s \approx 2^{\sqrt{\log n}}$.
2. Next we devise a black-box transformation to convert micro-structures into data structures for the intermediate case when input coordinates lie in a narrow $s \times n$ grid. Following Mortensen [21, 19], this part can be obtained from a $\sqrt{n}$-way divide-and-conquer similar to van Emde Boas trees [27]. (This part does not require bit packing.) The query and update time increase only by a $\log\log n$ factor as a result of this "van Emde Boas transformation".
3. Finally we give another black-box transformation to convert a narrow-grid data structure into a *macro-structure* for the general problem for large input. This part is obtained by using a global segment tree with fan-out near s. (This part does not require bit packing either.) The update time increases by a factor of $\log_s n$ (the height of the tree), which becomes near $\sqrt{\log n}$.

While this high-level plan may appear similar to our previous paper [12], at least two major difficulties arise, if we want the best update and query time: First, in part 1, buffered segment trees for 2-d orthogonal point location actually require $O((\log^2 s)/B) = O((\log^3 s)/w)$ update time, which forces us to set $s \approx 2^{\log^{1/3} n}$ and leads to a worse final update time near $\log_s n \approx \log^{2/3} n$. Second, the van Emde Boas transformation in part 2 causes at least one extra $\log\log n$ factor and leads to suboptimal query time in the end. A number of new ideas are thus needed to deal with these difficulties.

To overcome the first obstacle, our idea is to use micro-structures only for the simpler 1-sided case where input segments are horizontal rays, for which $O((\log s)/B)$ update cost is indeed possible. But how can we avoid using micro-structures for general 2-sided segments in part 3? We observe that an input segment appears more often as 1-sided than 2-sided at the nodes of segment tree, so we can afford to handle 2-sided updates by switching to a slower algorithm, with bootstrapping.

To overcome the second obstacle, we suggest a new van Emde Boas transformation to trade off the $\log\log n$ increase in the query time with a $\log^{\varepsilon} n$ increase in the update time. We can obtain such a trade-off only for the decision version of the problem, but luckily there are known randomized techniques [7] to reduce the original problem to the decision problem without hurting the expected query time.

As a by-product of our new van Emde Boas transformation, we can immediately obtain a data structure for *dynamic 1-d range emptiness* with $O(1)$ query time and $O(\log^{\varepsilon} N)$ update time for an integer universe bounded by N. This bound was known before: Mortensen, Pagh and Pătraşcu [22] in fact provided optimal results for a complete query/update time trade-off, but our solution in the constant query-time case is simpler, and may be of independent interest (if it has not been observed before).

Applications. Our result improves previous results even in various special cases:

- Dynamic *vertical ray stabbing* refers to the special case of vertical segment intersection reporting where the query segments are vertical rays. The problem has applications in databases, GIS, and networking. Previously, only logarithmic query and update time were known [26].

- Dynamic *vertical stabbing-min* refers to special case of vertical ray shooting where the query point has y-coordinate at $-\infty$ (stabbing-max is symmetric). Previously, Agarwal, Arge, and Yi at SODA'05 [1] obtained logarithmic query and update time. More recently, Nekrich [24] obtained $O\left(\frac{\log n}{\log\log n}\right)$ query and $O(\log n)$ update time; our $O\left(\log^{1/2+\varepsilon} n\right)$ update time is a significant improvement.

To further illustrate how fundamental our results are, we mention two offline applications (where as mentioned, the polynomially bounded universe assumption can automatically be ensured by sorting and replacing coordinates with ranks). Both applications are interesting in their own right.

- An $O\left(n\log^{1/2+\varepsilon} n\right)$-space data structure with $O(\log n)$ expected query time for static *3-d vertical ray shooting*: store a set of n axis-aligned rectangles in 3-d parallel to the xy-plane, so that we can find the lowest rectangle above a query point. This problem can be viewed as a variant of 3-d orthogonal point location. Our space bound is unusual and intriguing. The result can be immediately obtained by using the standard sweep-plane algorithm, together with a (partially) persistent version of our dynamic data structures for 2-d vertical ray shooting. The space usage is proportional to the total time to process n insertions and n deletions, which is $O\left(n\log^{1/2+\varepsilon} n\right)$; using Dietz's persistent arrays [14], the query time increases by a $\log\log n$ factor, to $O\left(\frac{\log n}{\log\log n}\log\log n\right) = O(\log n)$. This improves a previous $O\left(n\log^{1+\varepsilon} n\right)$-space data structure with $O(\log n)$ query time [8, Corollary 4.2(e)].
- A deterministic $O\left(n\frac{\log n}{\log\log n}\right)$-time algorithm for *single-source shortest paths in an unweighted intersection graph of n axis-aligned line segments* in 2-d, e.g., finding a path between two points with the minimum number of turns in an arrangement of vertical and horizontal line segments ("roads"). Recently, Chan and Skrepetos [11] described an $O(n\log n)$-time algorithm by simulating breadth-first search using a dynamic data structure for orthogonal segment intersection emptiness, performing at most n queries and n deletions. Our new data structure immediately improves the total running time to $O\left(n\log^{1/2+\varepsilon} n + n\frac{\log n}{\log\log n}\right) = O\left(n\frac{\log n}{\log\log n}\right)$.

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 segment. 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, or $O(\log\log n)$ in the polynomial universe case [27]).

We assume a word RAM model that allows for a constant number of non-standard 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 most of the paper, we focus on solving the decision problem, i.e., vertical segment intersection emptiness. Vertical ray shooting will be addressed in Section 7 afterwards. Adapting our algorithm for segment intersection reporting will be straightforward.

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

We now quickly review a few useful tools (also used in our previous paper [12]).

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 1** ([5, Lemma 4]). *In a weight-balanced B-tree of degree r, nodes at height i have weight $\Theta\left(r^i\right)$, and any sequence of n insertions requires at most $O\left(n/r^i\right)$ splits of nodes at height i.*

(We do not need to address balancing after deletions, since we can handle deletions lazily, and rebuild periodically when the size of the tree decreases or increases by a constant factor [5, 25].)

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 2** ([21, 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.*

Initial structure. For bootstrapping purposes, we need an initial data structure for vertical segment intersection emptiness with optimal $O\left(\log_w n\right)$ query time, allowing possibly large but polylogarithmic update time. Nekrich [23] has already given such a structure with $O\left(\log^{1+\varepsilon} n\right)$ update time. We state a weaker bound, which is sufficient for our purposes (and is simpler to obtain):

▶ **Lemma 3.** *For n horizontal segments in the plane, there exists a dynamic data structure for vertical segment intersection emptiness that support updates in $O\left(\log^{2+\varepsilon} n\right)$ amortized time and queries in $O\left(\log_w n\right)$ time.*

3 Micro-structures

In this section, we consider *micro-structures* for vertical segment intersection emptiness when the number of input segments s is small. This part relies on bit packing techniques. We focus on the *1-sided* special case, when all the input segments are horizontal rays. Without loss of generality, we may assume that all rays are rightward (since we can treat leftward rays separately). Vertical segment intersection emptiness in the 1-sided case is equivalent to 2-d *3-sided orthogonal range emptiness*: store a set of input points so that we can quickly decide whether a query 3-sided rectangle contains any input point. To see the equivalence, just replace the input rays with their endpoints, and each vertical query segment $\{x\} \times [y_1, y_2]$ with the 3-sided rectangle $(-\infty, x] \times [y_1, y_2]$.

We can adapt our previous micro-structures for 3-sided orthogonal range searching [12] to obtain:

▶ **Lemma 4.** *Let $b \geq 2$ be an integer. Given s horizontal (1-sided) rays with endpoints from $[s] \times \mathbb{R}$, there exists a dynamic data structure for vertical segment intersection emptiness with the following amortized update and query time:*

$$\begin{aligned} U_1(s) &= O\left(\frac{b \log^2 s}{w} + b \log^2 \log s\right) \\ Q_1(s) &= O\left(\log_b s\right). \end{aligned}$$

Proof. The case $b = 2$ has already been proved in [12, Lemmata 5(i) and 6]. We only briefly review the proof outline, to note the easy generalization to arbitrary b.

First consider the case when the endpoints all come from a static universe $[s] \times [s]$. The idea is to mimick an existing external-memory data structure with a block size of $B := \lceil \frac{\delta w}{\log s} \rceil$, observing that B points can be packed in a single word, assuming a sufficiently small constant δ. For such a external-memory structure, Chan and Tsakalidis [12] chose a *buffered* version [4] of a binary *priority search tree* ordered by y, citing Wilkinson [29]. Here, we use a buffered b*-ary* priority search tree instead, which according to Wilkinson [29, Lemma 1] has $O\left(b \cdot \frac{1}{B} \cdot \log s + 1\right) = O\left(\frac{b \log^2 s}{w} + 1\right)$ amortized update time and $O\left(\log_b s\right)$ amortized query time.

To make this data structure support a dynamic y-universe, Chan and Tsakalidis [12] applied monotone list labeling techniques; the extra update cost is $O(\log^2 \log s)$. It is straightforward to check that the same approach works in the b-ary variant, with an extra overhead factor of $O(b)$. ◀

Note that the first term in the above update time is constant when the number of segments s is bounded by $2^{\sqrt{w}}$.

We could also consider micro-structures for vertical segment intersection emptiness in the general (2-sided) case, but they seem to require more than two $\log s$ factors in the update time (not to mention possibly worse query time), which would result in a final update time worse than $\sqrt{\log n}$. Luckily, our macro-structures later need micro-structures only for the 1-sided case.

4 Van Emde Boas transformation

Next, we consider *macro-structures* for vertical segment intersection emptiness when the number of input segments n is large. As an intermediate step, we first adapt a technique of Mortensen [19, 21] that transforms any micro-structure for vertical segment intersection emptiness on s horizontal segments to one for n segments with endpoints from a narrow grid $[s] \times \mathbb{R}$.

The transformation uses a recursion similar to van Emde Boas trees [28], and increases both update and query time by a $\log \log n$ factor. Although the extra factor is small, we cannot afford to lose it if we want sublogarithmic query time at the end. We present a new variant of van Emde Boas recursion, with a parameter b, that allows us to trade off the query-time increase with an update-time increase:

▶ **Lemma 5.** *Let $b \geq 2$ be an integer. Given a dynamic data structure for vertical segment intersection emptiness on s horizontal segments with endpoints from $[s] \times \mathbb{R}$ achieving update time $U(s)$ and query time $Q(s)$, there exists a dynamic data structure for vertical segment intersection emptiness on n horizontal segments with endpoints from $[s] \times \mathbb{R}$ achieving the following update and query time:*

$$\begin{aligned} U(s,n) &= O\left(bU(s^5) \log^2 \log n + b \log^3 \log n\right) \\ Q(s,n) &= O\left(Q(s^5) \log_b \log n\right). \end{aligned}$$

An analogous transformation holds for the 1-sided special case of vertical segment intersection emptiness.

Proof. We present our variant of van Emde Boas recursion a little differently than usual, as a near-$n^{1/b}$-degree tree, with recursively defined secondary structures.

The data structure. We store a degree-r tree ordered by y, implemented as a weighted-balanced B-tree, for some parameter r to be chosen later. Each node corresponds to a horizontal slab; its slab is divided into its children's slabs by at most r dividing horizontal lines. We say that two input segments are in the same *class* if they have the same pair of left and right x-coordinates; there are at most s^2 classes. At each node v, we store the input segments in v's slab in one y-sorted list per class, in a colored predecessor search structure, and define the following lists:

1. Let $M(v)$ contain the bottommost and topmost segments (the "min" and the "max") in v's slab for each class. Since $|M(v)| \leq s^2$, we can maintain $M(v)$ in the given micro-structure supporting updates in $U(s^2)$ time and queries in $Q(s^2)$ time.
2. Let $R_0(v)$ contain the segments in v's slab after "rounding" down the y-coordinate to align with one of the r lines dividing the slab. Duplicates are removed from $R_0(v)$.
   Let $R(v)$ be equal to $R_0(v)$ but *excluding the bottommost and topmost rounded segment per class.* Since $R(v)$ has at most r distinct y-coordinates and at most $s^2 r$ segments, we can maintain $R(v)$ in a data structure supporting updates in $U(s, s^2 r)$ time and queries in $Q(s, s^2 r)$ time by recursion. (Note that we maintain $R(v)$, but not $R_0(v)$.) We further assume that this structure has $U_{\text{prep}}(s, s^2 r)$ *amortized preprocessing time*, i.e., preprocessing time divided by the number of input segments.

The update algorithm. To insert or delete a horizontal segment p, we proceed as follows:

1. Identify the path π of $O(\log_r n)$ nodes whose slabs contain p. Update the sorted lists for p's class at these nodes.
2. For each node $v \in \pi$ for which $M(v)$ changes, update the data structure for $M(v)$.
3. For each node $v \in \pi$ for which $R(v)$ changes, update the data structure for $R(v)$.

In step 1, the $O(\log_r n)$ sorted lists can be updated in $O\left(\log_r n \log^2 \log n\right)$ time by colored predecessor search (Lemma 2).

Step 2 takes at most $O(\log_r n)$ updates to the micro-structures and thus costs $O(\log_r n)\, U(s^2)$ time.

For step 3, we claim that $R(v)$ changes only at one node $v \in \pi$: specifically, the lowest node on π that contains at least one other segment of p's class. To see why, for any node $v' \in \pi$ strictly above v, there is no change to $R_0(v')$ (and thus $R(v')$) since there is another segment that gives the same rounded segment as p at v'; on the other hand, for any node $v' \in \pi$ strictly below v, there is no change to $R(v')$ because p is the only segment in its class at v' and would be excluded from $R(v')$.

Note that if $R(v)$ changes, it changes by at most a single insertion or deletion (for example, if the segment we are inserting becomes the new bottommost segment in a class, we insert the old bottommost segment to $R(v)$). Thus, step 3 takes $U(s, s^2 r)$ time.

To keep the tree balanced, we need to handle node splits. For nodes at height i, there are $O\left(n/r^i\right)$ splits by Lemma 1. A split at a non-leaf node v at height i requires rebuilding $M(v)$ and $R(v)$, which takes at most $O\left(r^i U(s^2) + r^i U_{\text{prep}}(s, s^2 r)\right)$ time. It also requires updating the y-coordinates of $O(s^2)$ segments in $R(v')$ at the parent v' of v, which takes $O\left(s^2\, U(s, s^2 r)\right)$ time. The total extra cost is at most

$$O\left(\sum_{i=1}^{\log_r n} (n/r^i) \cdot \left[r^i U(s^2) + r^i U_{\text{prep}}(s, s^2 r) + s^2\, U(s, s^2 r)\right]\right)$$
$$= O\left(n \cdot \left[U(s^2) \log_r n + U_{\text{prep}}(s, s^2 r) \log_r n + \frac{s^2}{r} U(s, s^2 r)\right]\right).$$

To summarize, we obtain the following recurrence for the amortized update time:

$$\begin{aligned} U(s,n) \;\leq\; & U\left(s, s^2 r\right) + O\left(\log_r n\right) U\left(s^2\right) + \\ & O\left(U_{\text{prep}}(s, s^2 r)\log_r n + \frac{s^2}{r} U(s, s^2 r) + \log_r n \log^2 \log n\right). \end{aligned}$$

The amortized preprocessing time is given by the following simpler recurrence, since balance is easily ensured at preprocessing:

$$U_{\text{prep}}(s,n) \;\leq\; U_{\text{prep}}\left(s, s^2 r\right) + O\left(\log_r n\right) U\left(s^2\right) + O\left(\log_r n \log^2 \log n\right).$$

The query algorithm. To answer a query for a vertical segment q with bottom endpoint q_1 and top endpoint q_2, we proceed as follows:

1. Find the lowest node v whose slab contains both q_1 and q_2 by performing an LCA query for the two leaves containing them.
2. Let v_1 and v_2 be the two children of v whose slabs contain q_1 and q_2.
3. Answer the query in $M(v_1)$, $M(v_2)$, and $M(v)$. Also, round q_1 upward and q_2 downward, then answer the query in $R(v)$. Return true iff one of these queries returns true.

To prove correctness, suppose that q intersects the horizontal input segment p. If p is in v_1's slab, then q intersects also the topmost segment in v_1 of p's class and so the query in $M(v_1)$ would return yes. If p is in v_2's slab, then similarly the query in $M(v_2)$ would return yes. If p is in neither slab, then q intersects the segment p after rounding and so the query in $R(v)$ would return yes, unless p after rounding is the topmost or bottommost rounded segment in v of its class. In this exceptional case, p would be excluded from $R(v)$, but then the query in $M(v)$ would return yes.

Since LCA queries take $O(1)$ time (with $O(1)$ update time) [13], we obtain the following recurrence for the query time:

$$Q(s,n) \;\leq\; Q\left(s, s^2 r\right) + O\left(Q\left(s^2\right)\right).$$

Conclusion. We set $r := s^2 n^{1/b}$ to obtain

$$\begin{aligned} U_{\text{prep}}(s,n) &\leq U_{\text{prep}}\left(s, s^4 n^{1/b}\right) + O\left(bU\left(s^2\right) + b\log^2 \log n\right) \\ U(s,n) &\leq \left(1 + \frac{1}{n^{1/b}}\right) U\left(s, s^4 n^{1/b}\right) + O\left(bU\left(s^2\right) + bU_{\text{prep}}(s,n) + b\log^2 \log n\right) \\ Q(s,n) &\leq Q\left(s, s^4 n^{1/b}\right) + O\left(Q\left(s^2\right)\right). \end{aligned}$$

For the base case, we can use $U_{\text{prep}}(s, s^5), U(s, s^5) = O(U(s^5))$ and $Q(s, s^5) = O(Q(s^5))$. The recurrences solve to $U_{\text{prep}}(s,n) = O\left(bU(s^5)\log\log n + b\log^2 \log n\right)$, $U(s,n) = O\left(b^2 U(s^5)\log^2 \log n + b^2 \log^3 \log n\right)$, and $Q(s,n) = O(Q(s^5)\log_b \log n)$. Resetting $b \leftarrow \lceil \sqrt{b} \rceil$ yields the lemma. ◀

Remark. The above approach gives a data structure for dynamic *1-d range emptiness* (which corresponds to the special case of $s = 1$) with $O(b \log\log N)$ update time and $O(\log_b \log N)$ query time in an integer universe $[N]$. (We do divide-and-conquer to reduce the universe size N rather than the number of points n; balancing is no longer an issue, so the extra $\log\log$ factor in the update bound goes away.) In particular, setting $b = \log^{\varepsilon} N$ gives $O(1)$ query time and $O(\log^{O(\varepsilon)} N)$ update time. This result was known before by Mortensen, Pagh, and Pătraşcu [22], who gave a more complicated method achieving a better (optimal) trade-off,

with $O(b \log\log N)$ update time and $O(\log\log_b \log N)$ query time, and also with $O(n)$ space. However, it is interesting to note that the above variant of van Emde Boas tree is sufficient for the constant-query-time case.

5 Segment tree transformation

We next describe macro-structures to transform data structures for vertical segment intersection emptiness for n segments in the narrow grid case, to the general case. The transformation is based on a multi-degree segment tree (the idea is standard and, for example, was used in Giyora and Kaplan's paper [16]).

▶ **Lemma 6.** *Given a dynamic data structure for vertical segment intersection emptiness on n horizontal segments with endpoints from $[s] \times \mathbb{R}$ achieving update time $U(s,n)$ and query time $Q(s,n)$, and a dynamic data structure for vertical segment intersection emptiness on n horizontal (1-sided) rays with endpoints from $[s] \times \mathbb{R}$ achieving update time $U_1(s,n)$ and query time $Q_1(s,n)$, there exists a data structure for dynamic vertical segment intersection emptiness on n horizontal segments with endpoints from $[N] \times \mathbb{R}$ achieving the following amortized update and query time:*

$$\begin{aligned} U(N,n) &= O\left(U_1(s,n)\log_s N + U(s,n) + \log_s N \log^2\log n\right) \\ Q(N,n) &= O\left((Q_1(s,n) + Q(s,n))\log_s N\right). \end{aligned}$$

Proof. We store a degree-s segment tree ordered by x, with N leaves and height $O(\log_s N)$; each node corresponds to a vertical slab.

We describe how each input segment p is stored. Let v be the lowest node that contains both endpoints of p, i.e., the LCA of the two leaves containing the endpoints. Let v_ℓ and v_r be the two children of v whose slabs contain the two endpoints. We divide p into three *subsegments*: the *left* and *right subsegments*, i.e., the parts of p within the slabs of v_ℓ and v_r respectively, and the remaining *middle subsegment*, i.e., the part within the union of the slabs of the children of v strictly between v_ℓ to v_r.

We store the middle subsegment of p in a data structure on the narrow grid $X_v \times \mathbb{R}$, where X_v is the set of x-coordinates of the $O(s)$ dividing vertical lines at node v.

We store the left subsegment of p along a path of $O(\log_s N)$ nodes. We first find the child v'_ℓ of v_ℓ whose slab contains the left endpoint of p. We divide the left subsegment into: the *left left subsegment*, i.e., the part within the slab of v'_ℓ, and the remaining *left middle subsegment*. We store the left middle subsegment in a 1-sided data structure on the narrow grid $X_{v'_\ell} \times \mathbb{R}$; note that this subsegment appears as a rightward ray in the narrow grid and so is indeed 1-sided. We then repeat for the left left subsegment in the slab at v'_ℓ.

We store the right subsegment of p symmetrically.

In addition, we store the y-coordinates of the segments/rays stored at each node v in a colored predecessor searching structure of Lemma 2, where segments/rays with endpoints in the same child's slab are assigned the same color. We also store the x-coordinates in another colored predecessor searching structure, where X_v is colored black and the rest is white.

To insert or delete a segment p, we insert or delete the middle subsegment in $O(U(s,n))$ time and the $O(\log_s N)$ pieces of the left/right subsegment in $O(U_1(s,n)\log_s N)$ time. Note that given the y-predecessor of the segment at a node v, we can obtain the y-predecessor/successor at the child by using the colored predecessor searching structure. We can also determine the x-predecessor/successor of its endpoints in X_v by another colored predecessor search. This takes total extra time $O\left(\log_s N \log^2\log n\right)$.

To answer an intersection emptiness query for a vertical segment q, we proceed down the path π of nodes whose slabs contain q, and perform queries in the narrow-grid structures (both general and 1-sided) at nodes in π. This takes $O((Q(s,n)+Q_1(s,n))\log_s N)$ time. ◀

6 Putting everything together

We can finally obtain our main result by combining with our preceding micro-structures and by bootstrapping.

▶ **Theorem 7.** *Given n horizontal segments with coordinates from $[N]\times\mathbb{R}$, there exists a dynamic data structure for vertical segment intersection emptiness that supports updates in amortized $O\left(\frac{\log N}{\log^{1/2-\varepsilon} n}+\log^{1/3} n\right)$ time and queries in $O\left(\frac{\log N}{\log\log n}\right)$ time if $N\geq n$.*

Proof. To make calculations more readable, we introduce the notation O^* to hide factors of the form $\log^{O(\varepsilon)} n$ and $w^{O(\varepsilon)}$.

Combining our van Emde Boas transformation in Lemma 5 with $b=\log^{\varepsilon} n$ and segment tree transformation in Lemma 6 gives

$$\begin{aligned} U(N,n) &= O^*\left(U_1(s)\log_s N+U(s)\right) \\ Q(N,n) &= O\left((Q_1(s)+Q(s))\log_s N\right). \end{aligned} \tag{1}$$

The 1-sided micro-structure in Lemma 4 with $b=w^{\varepsilon}$ gives $U_1(s^5)=O^*\left(\frac{\log^2 s}{w}+1\right)$ and $Q_1(s^5)=O(\log_w s)$. The initial structure in Lemma 3 gives $U(s^5)=O\left(\log^{2+\varepsilon} s\right)$ and $Q(s^5)=O(\log_w s)$. Thus,

$$\begin{aligned} U(N,n) &= O^*\left(\left(\frac{\log^2 s}{w}+1\right)\log_s N+\log^{2+\varepsilon} s\right) \\ Q(N,n) &= O(\log_w s\log_s N) = O(\log_w N). \end{aligned}$$

Setting $s=2^{\log^{1/3} N}$ then yields $U(N,n)=O^*\left(\log^{(2+\varepsilon)/3} N+\frac{\log^{4/3} N}{w}\right)$ and $Q(N,n)=O(\log_w N)$.

To improve the update time further, we bootstrap with our new bounds $U(s^5)=O^*\left(\log^{(2+\varepsilon)/3} s+\frac{\log^{4/3} s}{w}\right)$ and $Q(s^5)=O(\log_w s)$. Then

$$\begin{aligned} U(N,n) &= O^*\left(\left(\frac{\log^2 s}{w}+1\right)\log_s N+\log^{(2+\varepsilon)/3} s+\frac{\log^{4/3} s}{w}\right) \\ Q(N,n) &= O(\log_w s\log_s N) = O(\log_w N). \end{aligned} \tag{2}$$

Setting $s=2^{w^{1/2-\varepsilon}}$ yields $U(N,n)=O^*\left(\frac{\log N}{w^{1/2-\varepsilon}}+w^{1/3}\right)$ and $Q(N,n)=O(\log_w N)$. Setting the word size $w=\delta\log n$ gives the theorem. (The $\log^{1/3} n$ term could probably be lowered by further rounds of bootstrapping, but that term does not matter in the main case of interest, when $N=n^{O(1)}$.) ◀

7 Vertical ray shooting

We now extend our query algorithm for vertical segment intersection emptiness to vertical ray shooting.

Extended van Emde Boas transformation. First, we note a naive extension of the van Emde Boas transformation to support vertical ray shooting (the query time isn't optimized):

▶ **Lemma 8.** *The same data structure in Lemma 5 can answer vertical ray shooting queries in time* $\vec{Q}(s,n) = O\left(b\vec{Q}\left(s^5\right)\log\log n\right)$. *An analogous result holds for the 1-sided case.*

Proof. To answer a vertical ray shooting query for a point q, we proceed as follows:

1. Identify the path π of $O(\log_r n)$ nodes whose slabs contain q.
2. Find the lowest node $v \in \pi$ for which the answer of the query in $M(v)$ is nonempty.
3. Answer the query in $R(v)$ and in $M(v)$, and suppose that the two answers are in the slab of the children v_1 and v_2 of v respectively. Return the answer to the query in $M(v_1)$ or $M(v_2)$, whichever is lower.

To show correctness, let p be the lowest segment above q. After step 2, we know that p must be in the slab of v but not v's child in π. After step 3, we know that p must be in the slab of v_1 and in $M(v_1)$, unless p after rounding is the topmost or bottommost rounded segment in v of its class. In this exceptional case, p would be excluded from $R(v)$, but then p would be in the slab of v_2 and in $M(v_2)$.

We get the following recurrence for the query time:

$$\vec{Q}(s,n) \;\le\; \vec{Q}(s,s^2r) + O(\log_s n)\,\vec{Q}(s^2).$$

For $r = s^2n^{1/b}$, this gives $\vec{Q}(s,n) \le \vec{Q}(s,s^4n^{1/b}) + O\left(b\vec{Q}(s^2)\right)$, and the recurrence can be solved as before. ◀

Extended segment tree transformation. Next we extend the segment tree transformation. For this part, we will optimize the query time, using a randomized search technique from [7].

▶ **Lemma 9.** *In Lemma 6, if the given general and 1-sided data structures can answer vertical ray shooting queries in* $\vec{Q}(s,n)$ *and* $\vec{Q}_1(s,n)$ *time respectively, then the new data structure can answer vertical ray shooting queries in expected query time*

$$\vec{Q}(N,n) = O\left((Q_1(s,n)+Q(s,n))\log_s N + \left(\vec{Q}_1(s,n)+\vec{Q}(s,n)\right)\log\log_s N\right).$$

Proof. The technique [7] is based on the following simple well-known observation: the minimum of m unknown elements $y_1,\ldots,y_m$ can be found with m comparisons of the form "is $y_i < y$?" for a given value y, and $O(\log m)$ expected number of evaluations of the y_i's. (The observation follows by running the standard algorithm for the minimum, after randomly permuting the elements.)

To answer a vertical ray shooting query for a point q, we proceed down the path π of nodes whose slabs contain q, and apply the above observation with $m = O(\log_s N)$ and y_i representing the y-value of the lowest segment above q in the narrow-grid structures at the i-th node of π. Deciding "$y_i < y$?" is equivalent to performing a vertical segment emptiness query. The theorem follows. ◀

▶ **Theorem 10.** *The same data structure in Theorem 7 can answer vertical ray shooting queries in* $O\left(\frac{\log N}{\log\log n}\right)$ *time.*

Proof. By the above extensions, the combined van Emde Boas transformation (with $b = \log^\varepsilon n$) and segment tree transformation that yielded (1) has

$$\begin{aligned}\vec{Q}(N,n) &= O\left((Q_1(s)+Q(s))\log_s N + \left(\vec{Q}_1(s)+\vec{Q}(s)\right)\log^\varepsilon n\log\log n\log\log_s N\right)\\ &= O\left((Q_1(s)+Q(s))\log_s N + (Q_1(s)+Q(s))\log s\log^\varepsilon n\log\log n\log\log N\right),\end{aligned}$$

since a naive binary search gives $\vec{Q}_1(s) = O(Q_1(s) \log s)$ and $\vec{Q}(s) = O(Q(s) \log s)$. Then the structure in the final bootstrapping step that yielded (2) has

$$\vec{Q}(N, n) \quad = \quad O(\log_w s \log_s N + \log_w s \log s \log^{\varepsilon} n \log\log n \log\log N).$$

As $s = 2^{w^{1/2-\varepsilon}}$, we obtain $\vec{Q}(N, n) = O\left(\log_w N + w^{1-\Omega(\varepsilon)} \log\log N\right)$. As $w = \delta \log n$, we obtain the theorem. ◀

8 Future work

A number of interesting directions remain to be explored:

1. It would be nice to eliminate the assumption of polynomially bounded x-universe (i.e., the dependency in N). Our earlier paper [12] have already provided a mechanism to deal with a dynamic x-universe for the micro-structures in Lemma 4, but currently we have difficulty maintaining balance in the segment tree in Lemma 6 (standard weight-balanced B-trees seems to give an extra $\log_s n$ in the $U(s, n)$ term of the update time bound).
2. It would be nice to avoid randomization in our vertical ray shooting algorithm.
3. We have ignored space complexity throughout the paper. Many of the previous point location data structures achieves linear space. A naive upper bound on space for our data structure is n times the update time, i.e., $O\left(n \log^{1/2+\varepsilon} n\right)$. We believe that the bound can be lower by using more bit packing techniques, although it is unclear how to obtain linear space with our approach.
4. Insertion-only and deletion-only special cases are worth exploring. Here, $O\left(\frac{\log n}{\log\log n}\right)$ query time is not necessarily optimal; for example, see Wilkinson's insertion-only results on 2-d 3-sided orthogonal range searching [29]. As mentioned in the introduction, the deletion-only case has applications to geometric shortest paths [11].

References

1 Pankaj K. Agarwal, Lars Arge, Haim Kaplan, Eyal Molad, Robert E. Tarjan, and Ke Yi. An optimal dynamic data structure for stabbing-semigroup queries. *SIAM Journal on Computing*, 41(1):104–127, 2012. Preliminary version in SODA 2005. doi:10.1137/10078791X.

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, 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), 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 Guy E. Blelloch. Space-efficient dynamic orthogonal point location, segment intersection, and range reporting. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 894–903, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347180.

7 Timothy M. Chan. Geometric applications of a randomized optimization technique. *Discrete & Computational Geometry*, 22(4):547–567, 1999. doi:10.1007/PL00009478.

8 Timothy M. Chan. Persistent predecessor search and orthogonal point location on the word RAM. *ACM Transactions on Algorithms*, 9(3):22:1–22:22, 2013. Preliminary version in SODA 2011. doi:10.1145/2483699.2483702.

9 Timothy M. Chan and Yakov Nekrich. Towards an optimal method for dynamic planar point location. In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 390–409, 2015. doi:10.1109/FOCS.2015.31.

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 Timothy M. Chan and Dimitrios Skrepetos. All-pairs shortest paths in unit-disk graphs in slightly subquadratic time. In *Proceedings of the 27th International Symposium on Algorithms and Computation (ISAAC)*, pages 24:1–24:13, 2016. doi:10.4230/LIPIcs.ISAAC.2016.24.

12 Timothy M. Chan and Konstantinos Tsakalidis. Dynamic orthogonal range searching on the RAM, revisited. In *Proceedings of the 33rd International Symposium on Computational Geometry (SoCG)*, pages 28:1–28:13, 2017. doi:10.4230/LIPIcs.SoCG.2017.28.

13 Richard Cole and Ramesh Hariharan. Dynamic LCA queries on trees. *SIAM Journal on Computing*, 34(4):894–923, 2005. doi:10.1137/S0097539700370539.

14 Paul F. Dietz. Fully persistent arrays. In *Proceedings of the 1st Workshop for Algorithms and Data Structures (WADS)*, pages 67–74, 1989. doi:10.1007/3-540-51542-9_8.

15 Michael Fredman and Michael Saks. The cell probe complexity of dynamic data structures. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC)*, pages 345–354, 1989. doi:10.1145/73007.73040.

16 Yoav Giyora and Haim Kaplan. Optimal dynamic vertical ray shooting in rectilinear planar subdivisions. *ACM Transactions on Algorithms*, 5(3):28:1–28:51, 2009. Preliminary version in SODA 2007. doi:10.1145/1541885.1541889.

17 Katherine Jane Lai. Complexity of union-split-find problems. Master's thesis, MIT, 2008. URL: http://hdl.handle.net/1721.1/45638.

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

19 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.

20 Christian Worm Mortensen. *Data structures for orthogonal intersection searching and other problems*. PhD thesis, IT University of Copenhagen, 2006. URL: http://www.epust.dk/main.pdf?attredirects=0.

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

22 Christian Worm Mortensen, Rasmus Pagh, and Mihai Pătraşcu. On dynamic range reporting in one dimension. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 104–111, 2005. doi:10.1145/1060590.1060606.

23 Yakov Nekrich. Searching in dynamic catalogs on a tree. *CoRR*, abs/1007.3415, 2010. arXiv:1007.3415.

24 Yakov Nekrich. A dynamic stabbing-max data structure with sub-logarithmic query time. In *Proceedings of the 22nd International Symposium on Algorithms and Computation (ISAAC)*, pages 170–179, 2011. doi:10.1007/978-3-642-25591-5_19.

25 Mark H. Overmars. *The Design of Dynamic Data Structures*, volume 156 of *Lecture Notes in Computer Science*. Springer, 1983. doi:10.1007/BFb0014927.

26 Yufei Tao. Dynamic ray stabbing. *ACM Transactions on Algorithms*, 11(2):11:1–11:19, 2014. doi:10.1145/2559153.

27 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.

28 Peter van Emde Boas, Robert Kaas, and Erik Zijlstra. Design and implementation of an efficient priority queue. *Mathematical Systems Theory*, 10(1):99–127, 1976.

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