9-27-2022

ORTHOGONAL POINT LOCATION AND RECTANGLE STABBING QUERIES IN 3-D

Timothy M. Chan
University of Illinois Urbana-Champaign

Yakov Nekrich
Michigan Technological University, yakov@mtu.edu

Saladi Rahul
Indian Institute of Science

Konstantinos Tsakalidis
University of Liverpool

ORTHOGONAL POINT LOCATION AND RECTANGLE STABBING QUERIES IN 3-D *

Timothy M. Chan,[†] *Yakov Nekrich,*[‡] *Saladi Rahul,*[§] *and Konstantinos Tsakalidis*[¶]

ABSTRACT. In this work, we present a collection of new results on two fundamental problems in geometric data structures: *orthogonal point location* and *rectangle stabbing*.

- **Orthogonal point location.** We give the first linear-space data structure that supports 3-d point location queries on n disjoint axis-aligned boxes with optimal $O(\log n)$ query time in the (arithmetic) pointer machine model. This improves the previous $O\left(\log^{3/2} n\right)$ bound of Rahul [*SODA* 2015]. We similarly obtain the first linear-space data structure in the I/O model with optimal query cost, and also the first linear-space data structure in the word RAM model with sub-logarithmic query time. Our technique also improves upon the result of de Berg, van Kreveld, and Snoeyink [*Journal of Algorithms* 1995] for 3-d point location in (space filling) box subdivisions: we obtain a linear-space data structure with $O(\log^2 \log U)$ query time.
- **Rectangle stabbing.** We give the first linear-space data structure that supports 3-d 4-sided and 5-sided rectangle stabbing queries in optimal $O(\log_w n + k)$ time in the word RAM model, where k is the number of rectangles reported and w is the number of bits in a word. We similarly obtain the first optimal data structure for the closely related problem of 2-d top-k rectangle stabbing in the word RAM model, and also improved results for 3-d 6-sided rectangle stabbing.

For point location, our solution is simpler than previous methods, and is based on an interesting variant of the van Emde Boas recursion, applied in a round-robin fashion over the dimensions, combined with bit-packing techniques. For rectangle stabbing, our solution is a variant of Alstrup, Brodal, and Rauhe's grid-based recursive technique [*FOCS* 2000], combined with a number of new ideas.

1 Introduction

In this work we present a plethora of new results in 3-d on two fundamental problems in geometric data structures: (1) *orthogonal point location* (where the input rectangles or boxes

*A preliminary version of this paper appeared at *45th International Colloquium on Automata, Languages, and Programming (ICALP'18)*.

[†]*Department of Computer Science, University of Illinois at Urbana-Champaign*, tmc@illinois.edu. This research is partially supported by NSF Grant CCF-1814026.

[‡]*Department of Computer Science, Michigan Technological University*, yakov.nekrich@googlemail.com.
[§]*Department of Computer Science and Automation, Indian Institute of Science*, saladi@iisc.ac.in.
[¶]*Department of Computer Science, University of Liverpool*, K.Tsakalidis@liverpool.ac.uk.

are non-overlapping), and (2) *rectangle stabbing* (where the input rectangles or boxes can overlap).

1.1 Orthogonal point location

Point location is among the most central problems in the field of computational geometry, and is covered in textbooks and has countless applications. In this paper we study the *orthogonal point location* problem. Formally, we want to preprocess a set of n *disjoint* axis-aligned boxes (hyperrectangles) in $\mathbb{R}^d$ into a data structure, so that the box in the set containing a given query point (if any) can be reported efficiently. In each dimension, a box is closed on the left and open on the right. There are two natural versions of this problem, for (a) *arbitrary disjoint boxes* where the input boxes need not fill the entire space, and (b) a *box subdivision* where the input boxes fill the entire space.

Arbitrary disjoint boxes. Historically, the point location problem has been studied in the pointer machine model and the main question has been the following:

"Is there a linear-space structure with $O(\log n)$ query time?"

In 2-d this question has been successfully resolved: there exists a linear-space structure with $O(\log n)$ query time [31, 30, 19, 40, 43] (actually this result holds for nonorthogonal point location). In 3-d there has been work on this problem [20, 28, 2, 36], but the question has not yet been resolved. Even if $O(n \cdot \text{polylog}(n))$ space is allowed, achieving $O(\log n)$ query time is still a challenge. The currently best known result on the pointer machine model is a linear-space structure with $O(\log^{3/2} n)$ query time by Rahul [36]. In this paper,

- we obtain the first linear-space structure with $O(\log n)$ query time for 3-d orthogonal point location for arbitrary disjoint boxes. The structure works in the (arithmetic) pointer machine model and is *optimal* in this model.

The orthogonal point location problem has been studied in the I/O-model and the word RAM as well (please see Section 2.1 for a brief description of these models). In the I/O model, an optimal solution is known in 2-d [25, 8]: a linear-space structure with $O(\log_B n)$ query time, where B is the block size (this result holds for nonorthogonal point location). However, in 3-d the best known result is a linear-space structure with $O(\log_B^2 n)$ query I/O-cost by Nekrich [32] (for orthogonal point location for disjoint boxes).

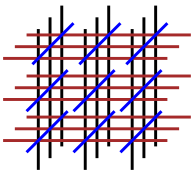
- In the I/O model, we obtain the first linear-space structure with $O(\log_B n)$ query cost for 3-d orthogonal point location for arbitrary disjoint boxes. This result is *optimal*.

In the word RAM model, an optimal solution in 2-d was given by Chan [12] with a query time of $O(\log\log U)$, assuming that input coordinates are in $[U] = \{0, 1, \ldots, U-1\}$. However, in 3-d the best known result for arbitrary disjoint boxes is a linear-space structure with $O(\log n \log\log n)$ query time: this result was not stated explicitly before but

can be obtained by an interval tree augmented with Chan's 2-d orthogonal point location structure [12] at each node. Our above new result with logarithmic query time is already an improvement even in the word RAM, but we can do slightly better still:

- In the w-bit word RAM model ($w \geq \max\{\log n, \log U\}$), we obtain the first linear-space structure with *sub-logarithmic* query time for 3-d orthogonal point location for arbitrary disjoint boxes. The time bound is $O(\log_w n)$. (We do not know if this result is optimal, however.)

Box subdivisions. In the plane, the two versions of the orthogonal point location problem are equivalent in the sense that any arbitrary set of n disjoint rectangles can be converted into a subdivision of $\Theta(n)$ rectangles via the vertical decomposition. In 3-d, the two versions are no longer equivalent, since there exist sets of n disjoint boxes that need $\Omega\left(n^{3/2}\right)$ disjoint boxes to fill the entire space. See the figure on the right (for details of this construction we refer to [27]).



In 3-d the special case of a box subdivision is potentially easier than the arbitrary disjoint boxes setting, as the former allows for a fast $O(\log^2 \log U)$ query time in the word RAM model with $O(n \log \log U)$ space, as shown by de Berg, van Kreveld, and Snoeyink [18] (with an improvement by Chan [12]).

- In the word RAM model, we further improve de Berg, van Kreveld, and Snoeyink's method to achieve a *linear*-space structure with $O(\log^2 \log U)$ query time for 3-d orthogonal point location on box subdivisions.

Our results for orthogonal point location in 3-d are summarized in Table 1.

Input	Model	Reference	Query time	Space	Remark
Arbitrary disjoint boxes	Pointer machine	[36]	$O(\log^{3/2} n)$	$O(n)$	
		New	$O(\log n)$	$O(n)$	optimal
	I/O	[32]	$O(\log_B^2 n)$	$O(n/B)$	
		New	$O(\log_B n)$	$O(n/B)$	optimal
	Word RAM	[12]+Interval tree	$O(\log n \log \log n)$	$O(n)$	
		New	$O(\log_w n)$	$O(n)$	–
Box subdivisions	Word RAM	[18]	$O(\log^2 \log U)$	$O(n \log \log U)$	
		New	$O(\log^2 \log U)$	$O(n)$	–

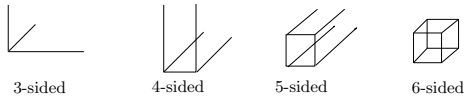
Table 1: A summary of our results for orthogonal point location in 3-d along with previously best known results.

1.2 Rectangle stabbing

Rectangle stabbing is a classical problem in geometric data structures [1, 3, 9, 15, 36], which is as old and as natural as orthogonal range searching. In orthogonal range searching, the input objects are points and the query objects are axis-aligned rectangles, whereas in rectangle stabbing we do the "inverse". Formally, we want to preprocess a set S of n axis-aligned boxes (possibly overlapping) in $\mathbb{R}^d$ into a data structure, so that the boxes in S containing a given query point q can be reported efficiently. (As one of many possible applications, imagine a flight booking website, where each user posts her desired dates of travel as a range $[D_1, D_2]$ and her desired departure time as a range $[T_1, T_2]$; then, for each flight with departure date x_q and departure time y_q, we would like to send notifications to users whose requirements match with this flight.)

In the word RAM model, Pătraşcu [34] gave a lower bound of $\Omega(\log_w n)$ query time for any data structure which occupies at most $n \log^{O(1)} n$ space to answer the 2-d rectangle stabbing query. Shi and Jaja [42] presented an optimal solution in 2-d which occupies linear space with $O(\log_w n + k)$ query time, where k is the number of rectangles reported.

We introduce some notation to define various types of rectangles in 3-d. A rectangle in 3-d is called $(3+t)$-*sided* if it is bounded in t out of the 3 dimensions and unbounded on one side in the remaining $3-t$ dimensions.



In the word RAM model, an optimal solution in 3-d is known *only* for the 3-sided rectangle stabbing query: a linear-space structure with $O(\log \log_w n + k)$ query time (by combining the work of Afshani [1] and Chan [12]; this is optimal due to the lower bound of Pătraşcu and Thorup [35]). Finding an optimal solution for 4-, 5-, and 6-sided rectangle stabbing has remained open.

3-d 4- and 5-sided rectangle stabbing. Currently, the best-known result for 4-sided and 5-sided rectangle stabbing queries by Rahul [36] occupies $O(n \log^* n)$ space with $O(\log n + k)$ and $O(\log n \log \log n + k)$ query time, respectively. This result holds in the pointer machine model. For 4-sided rectangle stabbing, adapting Rahul's solution to the word RAM model does not lead to any improvement in the query time (the bottleneck is in answering roughly $\log n$ 3-d 3-sided rectangle stabbing queries). For 5-sided rectangle stabbing, even if we assume the existence of an optimal 4-sided rectangle stabbing structure, plugging it into Rahul's solution can improve the query time to only $O(\log n + k)$, which is still suboptimal in the word RAM model. In this paper,

- we obtain the first *optimal* solution for 3-d 4-sided and 5-sided rectangle stabbing in the word RAM model: a linear-space structure with $O(\log_w n + k)$ query time.

2-d top-k rectangle stabbing. Recently, there has been a lot of interest in top-k range searching [4, 10, 11, 37, 38, 39, 41, 44]. Specifically, in the 2-d top-k rectangle stabbing problem, we want to preprocess a set of *weighted* axis-aligned rectangles in 2-d, so that

given a query point q and an integer k, the goal is to report the k largest-weight rectangles containing q. This problem is closely related to the 5-sided rectangle stabbing problem: by treating the weight as a third dimension, a rectangle r with weight $w(r)$ can be mapped to a 5-sided rectangle $r \times (-\infty, w(r)]$.

- By extending the solution for 3-d 5-sided rectangle stabbing problem, we obtain the first *optimal* solution for the 2-d top-k rectangle stabbing problem in the word RAM model: a linear-space structure with $O(\log_w n + k)$ query time.

3-d 6-sided rectangle stabbing. Our new solution to 3-d 5-sided rectangle stabbing, combined with standard interval trees, immediately implies a solution to 3-d 6-sided rectangle stabbing with a query time of $O(\log_w n \cdot \log n + k)$, which is already new (improving upon the $O(\log^2 n \cdot \log\log n + k)$ query time solution of Rahul [36]). But we can do slightly better still:

- We obtain a linear-space structure with $O(\log_w^2 n + k)$ query time for 3-d 6-sided rectangle stabbing problem in the word RAM model. We conjecture this to be optimal (an analogue is the lower bound of $\Omega(\log^2 n + k)$ query time for linear-space pointer machine structures [3]).

Back to orthogonal point location. Our solution for orthogonal point location in all the three models uses rectangle stabbing emptiness (deciding if any rectangle contains the query point) as a subroutine: informally, if there is an $S(n)$-space data structure with $Q(n)$ query time to solve the rectangle stabbing emptiness problem in $\mathbb{R}^d$, then one can obtain a data structure for orthogonal point location in $\mathbb{R}^{d+1}$ with $O(S(n))$-space and $O(Q(n))$ query time. By plugging in our new results for 3-d 6-sided rectangle stabbing, we obtain a linear-space word RAM structure which can answer any orthogonal point location query in 4-d in $O(\log_w^2 n)$ time, improving the previously known $O(\log^2 n \log\log n)$ bound [12].

Input	Reference	Query time	Space	Remark
4-sided	[36]	$O(\log n + k)$	$O(n \log^* n)$	
	New	$O(\log_w n + k)$	$O(n)$	optimal
5-sided	[36]	$O(\log n \log\log n + k)$	$O(n \log^* n)$	
	New	$O(\log_w n + k)$	$O(n)$	optimal
6-sided	[36]	$O(\log^2 n \log\log n + k)$	$O(n \log^* n)$	
	New	$O(\log_w^2 n + k)$	$O(n)$	–

Table 2: A summary of results for 3-d rectangle stabbing. The previous results hold in the pointer machine model, whereas our results hold in the word RAM model.

1.3 Our techniques

Our results are obtained using a number of new ideas (in addition to existing data structuring techniques), which we feel are as interesting as the results themselves.

3-d orthogonal point location. To better appreciate our new 3-d orthogonal point location method, we first recall that the current best word-RAM method had $O(\log n \log\log n)$ query time [12], and was obtained by building an interval tree over the x-coordinates, and at each node of the tree, storing Chan's 2-d point location data structure [12] on the yz-projection of the rectangles. Interval trees caused the query time to increase by a logarithmic factor, while Chan's 2-d structures achieved $O(\log\log n)$ query time via a complicated van-Emde-Boas-like recursion. We can thus summarize this approach loosely by the following recurrence for the query time (superscripts refer to the dimension):

$$Q^{(3)}(n) = O(Q^{(2)}(n)\log n) \text{ and } Q^{(2)}(n) = Q^{(2)}(\sqrt{n}) + O(1)$$
$$\Longrightarrow Q^{(3)}(n) = O(\log n \log\log n).$$

Note that naively increasing the fan-out of the interval tree to, say, f could reduce the query time to $O(\log_f n \cdot \log\log n)$ but would blow up the space usage to $O(nf)$.

In the pointer machine model, the current best data structure by Rahul [36], with $O(\log^{3/2} n)$ query time, required an even more complicated combination of interval trees, Clarkson and Shor's random sampling technique, 3-d rectangle stabbing, and 2-d orthogonal point location.

To avoid the extra $\log\log n$ factor, we cannot afford to use Chan's 2-d orthogonal point location structure as a subroutine; and we cannot work with just yz-projections, which intuitively cause loss of efficiency. Instead, we propose a more direct solution based on a new van-Emde-Boas-like recursion, aiming for a new recurrence of the form

$$Q^{(3)}(n) = Q^{(3)}(\sqrt{n}) + O(\log n).$$

The $O(\log n)$ term arises from the need to solve 2-d rectangle stabbing subproblems, on projections along all three directions (the yz-, xz-, and xy-plane), applied in a *round-robin* fashion. The new recurrence then solves to $O(\log n)$—notice how $\log\log$ disappears, unlike the usual van Emde Boas recursion. In the word RAM model, we can even use known sub-logarithmic solutions to 2-d rectangle stabbing to get $O(\log_w n)$ query time.

We emphasize that our new method is much *simpler* than the previous, slower methods, and is essentially self-contained except for the use of a known data structure for 2-d rectangle stabbing emptiness (which reduces to standard 2-d orthogonal range counting).

One remaining issue is space. In our new method, a rectangle is stored $O(\log\log n)$ times, due to the depth of the recursion. To achieve linear space, we need another idea, namely, *bit-packing* tricks, to compress the data structure. Because of the rapid reduction of the universe size in the round-robin van-Emde-Boas recursion, the amortized space in words per input box satisfies a recurrence of the form

$$s(n) = s(\sqrt{n}) + O\left(\frac{\log n}{w}\right) \implies s(n) = O\left(\frac{\log n}{w}\right) = O(1).$$

Our new result on the subdivision case is obtained by a similar space-reduction trick.

3-d 5-sided rectangle stabbing. For 3-d rectangle stabbing, the previous solution by Rahul [36] was based on a grid-based, $\sqrt{n}$-way recursive approach of Alstrup, Brodal, and Rauhe [7], originally designed for 2-d orthogonal range searching. The fact that the approach can be adapted here is nontrivial and interesting, since our input objects are now more complicated (rectangles instead of points) and the target query time is quite different (near logarithmic rather than $\log\log$). More specifically, Rahul first solved the 4-sided case via a complicated data structure, and then applied Alstrup et al.'s technique to reduce 5-sided rectangles to 4-sided rectangles, which led to a query-time recurrence similar to the following (subscripts denote the number of sides, and output cost related to k is ignored):

$$Q_4(n) = O(\log n) \text{ and } Q_5(n) = 2Q_5(\sqrt{n}) + O(Q_4(n)) \implies Q_5(n) = O(\log n \log\log n).$$

Intuitively, the reduction from the 5-sided to the 4-sided case causes loss of efficiency. To avoid the extra $\log\log n$ factor, we propose a new method that is also based on Alstrup *et al.*'s recursive technique, but reduces 5-sided rectangles directly to 3-sided rectangles, aiming for a new recurrence of the form

$$Q_3(n) = O(\log\log_w n) \text{ and } Q_5(n) = 2Q_5(\sqrt{n}) + O(Q_3(n)).$$

During recursion, we do not put 4-sided rectangles in separate structures (which would slow down querying), but instead use a common tree for both 4-sided and 5-sided rectangles. With the base case $Q_5(\log^{1/4} n) = O(1)$, the new recurrence then solves to $Q_5(n) = O(\log_w n)$—notice how $\log\log$ again disappears, and notice how this gives a new result even for the 4-sided case.

One remaining issue is space. Again, we can compress the data structure by incorporating bit-packing tricks (which was also used in Alstrup *et al.*'s original method). For 4- and 5-sided rectangle stabbing, the space recurrence then solves to linear.

However, with space compression, a new issue arises. The cost of reporting each output rectangle in a query increases to $O(\log\log n)$ (the depth of the recursion), because we need to *decode* the label of a compressed rectangle to obtain its original coordinates. In other words, the query cost becomes $O(\log_w n + k\log\log n)$ instead of $O(\log_w n + k)$. This extra decoding overhead also occurred in previous work on 2-d orthogonal range searching by Alstrup *et al.* [7] and Chan *et al.* [13], and it is open how to avoid the overhead for that problem without sacrificing space (this is related to the so-called *ball inheritance problem* [13]).

We observe that for the 4- and 5-sided rectangle stabbing problem, a surprisingly simple idea suffices to avoid the overhead: instead of keeping pointers between consecutive levels of the recursion tree, we just keep pointers directly from each level to the *leaf* level.

3-d 6-sided rectangle stabbing. We can solve 6-sided rectangle stabbing by using our result for 5-sided rectangle stabbing as a subroutine. However, the naive reduction via interval trees increases the query time by a $\log n$ factor instead of $\log_w n$. To speed up

querying, the standard idea is to use a tree with a larger fan-out w^ε. This leads to various colored generalizations of 2-d rectangle stabbing with a small number w^ε of colors. Much of our ideas can be extended to solve these colored subproblems in a straightforward way, but a key subproblem, of answering colored 2-d dominance searching queries in $O(\log\log_w n+k)$ time with linear space, is nontrivial. We solve this key subproblem via a clever use of shallow cuttings [5], combined with a grouping trick, which may be of independent interest.

2 Orthogonal Point Location in 3-d

2.1 Preliminaries

Our solution to 3-d orthogonal point location will require known data structures for *2-d orthogonal point location* and *2-d rectangle stabbing emptiness* (deciding if any rectangle contains the query point). We start by briefly describing the three models of computation for which we obtain our results.

On the Models. Throughout this paper, the *pointer machine model* refers to the "arithmetic pointer machine" (APM) in the terminology from Chazelle's paper [16]: Each word (or memory cell) stores a constant number of pointers, input rectangles, and/or w-bit integers for a fixed w. We support pointer chasing (i.e., following a pointer to an entry within the same word) and standard arithmetic operations (addition, subtraction, multiplication, division, shift), and comparisons on w-bit integers in unit time each, but do not allow pointer arithmetic. It is assumed that $w \geq \log n$ (which is reasonable since a pointer requires $\Omega(\log n)$ bits) and that $w \geq \log U$ (since the coordinates of an input rectangle requires $\Omega(\log U)$ bits).

In the *I/O model* [6], each block is assumed to hold B words for a fixed B, where each word stores the coordinate of an input rectangle or a w-bit integer, again assuming that $w \geq \max\{\log n, \log U\}$. We support block reads/writes with unit cost each; all other operations on a block are free.

In the *word RAM model* [22], each word stores a w-bit integer, again assuming that $w \geq \max\{\log n, \log U\}$; we support standard arithmetic operations (additions, subtractions, and multiplications), comparisons, bitwise logical operations, and shifts on w-bit integers in unit time each, and allow these w-bit integers to be used as pointers. Furthermore, it is assumed that the coordinates of the input rectangles are w-bit integers.

Subroutines. We will need data structures for 2-d point location and 2-d rectangle stabbing emptiness as subroutines for our solution.

Lemma 1. *Given n disjoint axis-aligned rectangles in $[U]^2$ ($n \leq U \leq 2^w$), there are data structures for point location using $O\left(\frac{n\log U}{w}\right)$ words of space and*

- *$O(\log n)$ query time in the pointer machine model;*
- *$O(\log_B n)$ query cost in the I/O model;*
- *$O(\min\{\log\log U, \log_w n\})$ query time in the word RAM model.*

Lemma 2. *Given n (possibly overlapping) axis-aligned rectangles in $[U]^2$ ($n \leq U \leq 2^w$), there are data structures for rectangle stabbing emptiness using $O\left(\frac{n \log U}{w}\right)$ words of space and*

- *$O(\log n)$ query time in the pointer machine model;*
- *$O(\log_B n)$ query cost in the I/O model;*
- *$O(\log_w n)$ query time in the word RAM model.*

Proof of Lemmata 1 and 2. For Lemma 1, such data structures for 2-d orthogonal point location can be found in [31, 30, 19, 40, 43] for the pointer machine model, [25, 8] for the I/O model, and [12] for the word RAM model. For Lemma 2, 2-d rectangle stabbing emptiness (or more generally, rectangle stabbing counting) is known to be reducible to 2-d orthogonal range counting [21], and such data structures for 2-d orthogonal range counting can be found in [16] for the pointer machine model, [26] for the I/O model, and [29] for the word RAM model.

All these known data structures technically require $O(n)$ words of space, or more precisely, $O(n \log U)$ bits of space. In the I/O model or word RAM model, we can easily pack the data structures in $O\left(\frac{n \log U}{w}\right)$ words of space without increasing the query cost when $\log U \ll w$. In the pointer machine model, we may not be able to pack the data structures in general, since we are not allowed pointer arithmetic. Nevertheless, it is not difficult to modify the existing data structures to achieve the compressed space bound. Next, we present the technical details of the modifications needed.

Proof of Lemma 1 for pointer machines. For 2-d orthogonal point location, one solution is via $(1/r)$-*cuttings* [24]: we can partition the plane into $O(r)$ disjoint rectangular cells, each intersecting $O(n/r)$ line segments (edges of the input rectangles), where we choose $r = \frac{n \log U}{\delta w}$ for a sufficiently small constant δ.

We build a point location structure [31, 30, 19, 40, 43] for the $O(r)$ cells with $O(\log r)$ query time in the pointer machine model; the space usage of this structure in words is $O(r)$, which is within the allowed bound $O\left(\frac{n \log U}{w}\right)$, so there is no need for bit packing here.

For each cell, we store the $O(n/r)$ line segments in a point location structure [30] with $O(\log(n/r))$ query time; the space usage of this structure in bits is $O((n/r) \log U)$, which is $O(w\delta)$, so the entire structure can be packed in a single word. Although pointer chasing is not directly supported in the pointer machine model when multiple "micro-pointers" are packed in a word, we can simulate each pointer chasing step here in constant time (a pointer can be extracted with one shift and one bitwise AND operation).

Given a query point q, we can first find the cell containing q in $O(\log r)$ time and then finish the query inside the cell in $O(\log(n/r))$ time. The overall query time is $O(\log n)$.

Proof of Lemma 2 for pointer machines. Rectangle stabbing emptiness in 2-d reduces to dominance range counting in 2-d [21]. Chazelle's *compressed range tree* structure [16] solves the latter problem with $O(n)$ words of space and $O(\log n)$ time in the pointer machine model. We observe that his data structure actually achieves $O\left(\frac{n \log U}{w}\right)$ words of space, after minor

modifications which are described next.

At each level of the range tree, Chazelle's structure stores lists consisting of a total of $O\left(\frac{n}{w}\right)$ words ($O\left(\frac{n}{w}\right)$ w-bit integers as well as $O\left(\frac{n}{w}\right)$ pointers to words in lists at the next level). The total number of words over all levels of the tree is $O\left(\frac{n \log n}{w}\right) = O\left(\frac{n \log U}{w}\right)$.

We decrease the height of the range tree by making the leaf nodes contain b points, where we choose $b = \frac{\delta w}{\log U}$ for a sufficiently small constant δ. This way, the number of nodes in the tree itself is $O(n/b) = O\left(\frac{n \log U}{w}\right)$. Inside each leaf, we store the b points in another instance of Chazelle's structure; the space usage of this structure in bits is $O(b \log U)$, which is $O(\delta w)$, so the entire structure can be packed in a single word. Again, we can simulate each pointer chasing step here in constant time by arithmetic operations and shifts within the word.

To answer a dominance range counting query, we descend along a path in the compressed range tree, which requires $O(\log(n/b))$ time by following pointers in the lists stored at the path and doing various arithmetic operations and shifts on w-bit integers. At the leaf of the path, we can finish the query in $O(\log b)$ time. The overall query time is $O(\log n)$.

2.2 Our solution

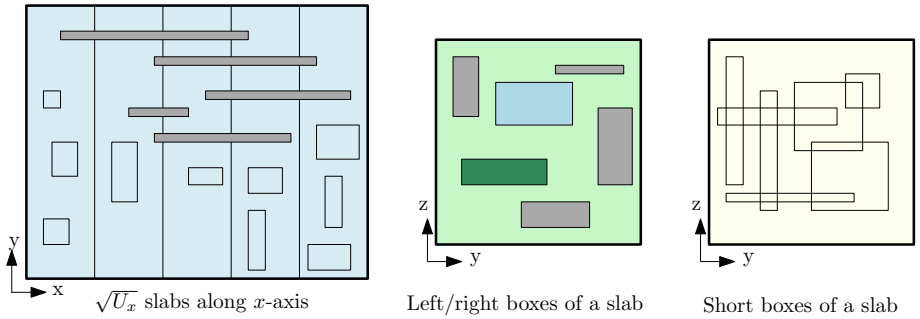
Data structure. We are now ready to describe our data structure for 3-d orthogonal point location. We focus on the pointer machine model first. At the beginning, we apply a rank space reduction so that all coordinates are on the integer grid $[4n]^3$, where n is the global number of input boxes (map input coordinates to their ranks; rank-i coordinate is mapped to $2i$ on the integer grid). Given a query point, we can initially find the ranks of its coordinates by three predecessor searches (costing $O(\log n)$ time in the pointer machine model).

We describe our preprocessing algorithm recursively. The input to the preprocessing algorithm is a set of n disjoint boxes that are assumed to be aligned to the $[U_x] \times [U_y] \times [U_z]$ grid. At the beginning, $U_x = U_y = U_z = 4n$.

Without loss of generality, assume that $U_x \geq U_y, U_z$. We partition the $[U_x] \times [U_y] \times [U_z]$ grid into $\sqrt{U_x}$ equal-sized vertical slabs perpendicular to the x-direction. See Figure 1. In the symmetric case $U_y \geq U_x, U_z$ or $U_z \geq U_x, U_y$, we partition along the y- or z-direction instead. We classify the boxes into two categories:

- *Short boxes.* For each slab, define its short boxes to be those that lie completely inside the slab.
- *Long boxes.* Long boxes intersect the boundary (vertical plane) of at least one slab. Each long box $\mathcal{B}$ is broken into three disjoint boxes:
  - *Left box.* Let s_L be the slab containing the left endpoint (with respect to the x-axis) of $\mathcal{B}$. The left box is defined as $\mathcal{B} \cap s_L$.
  - *Right box.* Let s_R be the slab containing the right endpoint of $\mathcal{B}$. The right box is defined as $\mathcal{B} \cap s_R$.

– *Middle box.* The remaining portion of box $\mathcal{B}$ after removing its left and right box, i.e. $\mathcal{B} \setminus (s_L \cup s_R)$.



$\sqrt{U_x}$ slabs along x-axis (a) Left/right boxes of a slab (b) Short boxes of a slab (c)

Figure 1: Boxes obtained after partitioning along the x-direction.

We build our data structure as follows:

1. *Planar point location structure.* For each slab, we project its left boxes onto the yz-plane. The projected boxes remain disjoint, since they intersect a common boundary. We store them in a data structure for 2-d orthogonal point location by Lemma 1. We do this for the slab's right boxes as well.

2. *Rectangle stabbing structure.* For each slab, we project its short boxes onto the yz-plane. The short boxes are not necessarily disjoint. We store them in a data structure for 2-d rectangle stabbing emptiness by Lemma 2.

3. *Recursive middle structure.* We recursively build a *middle structure* on all the middle boxes.

4. *Recursive short structures.* For each slab, we recursively build a *short structure* on all the short boxes inside the slab.

By translation or scaling, these recursive short structures or middle structure can be made aligned to the $\lceil\sqrt{U_x}\rceil \times [U_y] \times [U_z]$ grid. Observe that our construction ensures that each box is stored in at most one recursive structure. In addition, we store a list of pairs, where the first entry in each pair is one of the left/right/middle boxes and the second entry is the corresponding original box. We pack these pairs in $O\left(\frac{n \log(U_x U_y U_z)}{w}\right)$ words and then build a predecessor data structure on the pairs.

Query algorithm. The following lemma is crucial for deciding whether to query recursively the middle or the short structure.

Lemma 3. *Given a query point (q_x, q_y, q_z), if the query with (q_y, q_z) on the rectangle stabbing emptiness structure of the slab that contains q_x returns*

- NON-EMPTY, *then the query point cannot lie inside a box stored in the middle structure.*
- EMPTY, *then the query point cannot lie inside a box stored in the slab's short structure.*

Proof. If NON-EMPTY is returned, then the query point is stabbed by the extension (along the x-direction) of a box in the slab's short structure and cannot be stabbed by any box stored in the middle structure, because of disjointness of the input boxes. If EMPTY is returned, then obviously the query point cannot lie inside a box stored in the short structure. □

To answer a query for a given point (q_x, q_y, q_z), we proceed as follows:

1. Find the slab that contains q_x by predecessor search over the slab boundaries.
2. Query with (q_y, q_z) the planar point location structures at this slab. If a left or a right box returned by the query contains the query point, then we are done.
3. Query with (q_y, q_z) the rectangle stabbing emptiness structure at this slab. If it returns NON-EMPTY, query recursively the slab's short structure, else query recursively the middle structure (after appropriate translation/scaling of the query point).

In step 3, to decode the coordinates of the output box, we need to map from a left/right/middle box to its original box; this can be done naively by a predecessor search in the list of pairs we have stored.

Query time analysis. Let $Q(U_x, U_y, U_z)$ denote the query time for our data structure in the $[U_x] \times [U_y] \times [U_z]$ grid. Observe that n, the number of boxes, is trivially upper-bounded by $U_xU_yU_z$ because of disjointness. The predecessor search in step 1, the 2-d point location query in step 2, and the 2-d rectangle stabbing query in step 3 all take $O(\log n) = O(\log(U_xU_yU_z))$ time by Lemmata 1 and 2. We thus obtain the following recurrence, assuming that $U_x \geq U_y, U_z$:

$$Q(U_x, U_y, U_z) = Q\left(\sqrt{U_x}, U_y, U_z\right) + O(\log(U_xU_yU_z)).$$

If $U_x = U_y = U_z = U$, then three rounds of recursion will partition along the x-, y-, and z-directions and decrease U_x, U_y, and U_z in a round-robin fashion, yielding

$$Q(U, U, U) = Q\left(\sqrt{U}, \sqrt{U}, \sqrt{U}\right) + O(\log U),$$

which solves to $Q(U, U, U) = O(\log U)$. As $U = 4n$ initially, we get $O(\log n)$ query time.

Space analysis. Let $s(U_x, U_y, U_z)$ denote the *amortized* number of words of space needed per input box for our data structure in the $[U_x] \times [U_y] \times [U_z]$ grid. The amortized number of words per input box for the 2-d point location and rectangle stabbing structures is $O\left(\frac{\log(U_x U_y U_z)}{w}\right)$ by Lemmata 1 and 2. We thus obtain the following recurrence, assuming that $U_x \geq U_y, U_z$:

$$s(U_x, U_y, U_z) = s\left(\sqrt{U_x}, U_y, U_z\right) + O\left(\frac{\log(U_x U_y U_z)}{w}\right).$$

Three rounds of recursion yield

$$s(U, U, U) = s\left(\sqrt{U}, \sqrt{U}, \sqrt{U}\right) + O\left(\frac{\log U}{w}\right),$$

which solves to $s(U, U, U) = O\left(\frac{\log U}{w}\right)$. As $U = 4n$ initially, the total space in words is $O\left(n\frac{\log n}{w}\right) = O(n)$. Note that the above analysis ignores an overhead of $O(1)$ words of space per node of the recursion tree, but by shortcutting degree-1 nodes, we can bound the number of nodes in the recursion tree by $O(n)$. To summarize, we claim the following results:

Theorem 1. *Given n disjoint axis-aligned boxes in 3-d, there are data structures for point location with $O(n)$ words of space and $O(\log n)$ query time in the pointer machine model, $O(\log_B n)$ query cost in the I/O model, and $O(\log_w n)$ query time in the word RAM model.*

Proof. The proof for the pointer machine model has already been presented above. Now we present proofs for the I/O model and the word RAM model. In the I/O model, the analysis is similar, with a modified recurrence for the query cost:

$$Q(U, U, U) = Q\left(\sqrt{U}, \sqrt{U}, \sqrt{U}\right) + O(\log_B U).$$

For the base case $U \leq B^{1/3}$, we have $Q(U, U, U) = O(1)$ trivially, since $n \leq U^3 \leq B$. Solving the recurrence yields $O(\log_B n)$ query cost. The space usage remains $O(n)$ words (i.e., $O(n/B)$ blocks).

In the word RAM model, the analysis is again similar, with

$$Q(U, U, U) = Q\left(\sqrt{U}, \sqrt{U}, \sqrt{U}\right) + O(\log_w U).$$

For the base case $U \leq w$, we have $Q(U, U, U) = O(1)$ by switching to another known method: Orthogonal point location in 3-d reduces to 6-d dominance emptiness, for which there is a known method [14] with $O\left(n(\log_w n)^4\right)$ words of space and $O\left((\log_w n)^5\right)$ query time in the word RAM. (The method in [14] can be modified to report a witness if the range is non-empty.) Since $n \leq U^3 \leq w^3$, we have $\log_w n = O(1)$, and so the space bound is $O(n)$ and query bound is $O(1)$ for the base case. Solving the recurrence yields $O(\log_w n)$ query time. □

2.3 Other consequences

Orthogonal point location in 4-d. We will extend the above approach to answer orthogonal point location in 4-d in the word RAM model. As subroutines, we will now plug in the bounds for 3-d 6-sided rectangle stabbing (Theorem 4) and the bounds for 3-d orthogonal point location (Theorem 1) into our framework. The analysis is again similar, with

$$Q\left(U,U,U,U\right) = Q\left(\sqrt{U},\sqrt{U},\sqrt{U},\sqrt{U}\right) + O\left(\log_w^2 U\right),$$

which solves to $O(\log_w^2 U) = O(\log_w^2 n)$. The base case of $U \leq w$ can be handled in constant time by reducing it to 8-d dominance emptiness query. The space continues to be linear, since the analysis in Section 2.2 shows that as long as the subroutines occupy linear space, our approach will blow up the space by only a constant factor. Thus, we obtain a linear-space structure which can answer a 4-d orthogonal point location query in $O(\log_w^2 n)$ time. This improves upon the previously best known result with query time $O(\log^2 n \cdot \log\log n)$ [12].

Higher dimensions (beyond 4-d). The same approach can be extended to higher dimensions, reducing the complexity of d-dimensional orthogonal point location to that of $(d-1)$-dimensional rectangle stabbing emptiness. However, known data structures for higher-dimensional rectangle stabbing [3] require superlinear space, whereas the simpler approach mentioned in the introduction, of using interval trees to reduce the dimension, gives $O\left(\log^{d-2} n\right)$ query time while keeping linear space in the pointer machine model. This improves upon the previously best known result by Rahul [36] which uses linear space and answers a query in $O(\log^{d-3/2} n)$ time in the pointer machine model.

The case of 3-d subdivisions. Our approach can also be used to improve the space bound of de Berg, van Kreveld, and Snoeyink's point location structure [18] for 3-d orthogonal subdivisions, from $O\left(n \log\log U\right)$ space to $O\left(n\right)$, in the word RAM model.

Theorem 2. *Given a subdivision formed by n disjoint (space-filling) axis-aligned boxes in 3-d, there is a data structure for point location with $O\left(n\right)$ words of space and $O\left(\log^2 \log U\right)$ query time in the word RAM model.*

Proof. De Berg et al.'s method [18, Theorem 2.4] was already based on a van Emde Boas recursion, partitioning along the x-direction. Unlike our data structure in Section 2.2 which has four components, their solution was able to avoid using the rectangle stabbing structure by exploiting the fact that the input is a subdivision (they used the remaining three components). For the planar point location structure, vertical decomposition of the rectangles (left and right boxes projected onto the yz-plane) was performed which filled the entire yz-plane and increased the number of rectangles by only a constant factor. As a result, a data structure for 2-d point location for rectangular subdivisions was used, which occupies linear space and can answer a query in $O(\log\log U)$ time [12]. The space occupied by their data structure was $O(n \log\log U)$ since each box was stored at $O(\log\log U)$ nodes (due to the recursive middle structures).

To answer a query, they performed only step 1 and step 2 of our query algorithm (in Section 2.2) to decide whether to recurse on the middle structure or the short structure. In step 2 of our query algorithm, let r be the rectangle containing (q_y, q_z). Three cases arise in the case of a subdivison: (a) if r corresponds to a box in our input and contains q, then we are done, (b) if r corresponds to a box in our input and does not contain q, then q lies inside a short box, and (c) if r does not correspond to a box in our input, then q lies inside a middle box.

Our new change is to do the van Emde Boas recursion not just along the x-direction but along all three axis directions in a round-robin fashion. This leads to the same recurrence for space as in Section 2.2, and hence, improves the space to $O(n)$ words. The query time satisfies the following recurrence:

$$Q(U,U,U) = Q\left(\sqrt{U},\sqrt{U},\sqrt{U}\right) + O(\log\log U),$$

which solves to $O(\log^2 \log U)$. □

3 3-d 5-sided Rectangle Stabbing

In this section we will present an optimal solution to 3-d 5-sided rectangle stabbing problem in the word RAM model. The 5-sided rectangles are unbounded on one side along z-axis.

Preliminaries. We will use the following two lemmata (a slow structure and a small-sized structure) to construct the final solution.

Lemma 4. *(Rahul [36].) There is a data structure of size $O(n)$ words that can answer a 5-sided 3-d rectangle stabbing query in $O(\log^2 n \cdot \log\log n + k)$ time.*

Lemma 5. *(Leaf structure.) Consider a set of $m = O(w^{1/4})$ 5-sided 3-d rectangles lying on a grid $[2m] \times [2m] \times [m]$. Then there is a data structure of size $O(m)$ words that can answer a 5-sided 3-d rectangle stabbing query in $O(1+k)$ time.*

Proof. Two queries are considered combinatorially different if the output of both the queries are different. For 1-d rectangle stabbing, the number of combinatorially different queries will be $O(m)$: starting from the leftmost left-endpoint in the set, start walking towards the right and notice that the output changes only when we arrive at a left endpoint or a right endpoint in the set. With some thought, it can be observed that the number of combinatorially different queries for d-dimensional rectangle stabbing is $O(m^d)$.

To prove the lemma, the idea is to *explicitly* store the output of every possible query on this dataset. Notice that there are only $O(m^3)$ combinatorially different queries. Also, the description of each rectangle requires $\Theta(\log m)$ bits. Therefore, the space in bits occupied by explicitly storing the output of each query will be $O(m^3) \times O(m) \times O(\log m) = O(m^5) = O(w^{5/4})$. The space in words will be $O(w^{1/4}) = O(m)$. A three-dimensional array can be used to store these results, so that a query point can be located in the three-dimensional array in constant time and then the output can be reported in $O(k)$ time. □

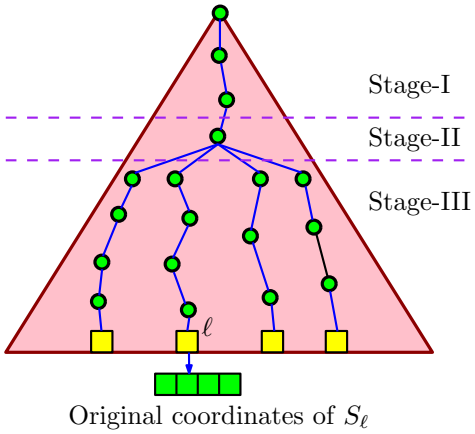


Figure 2: Tree $\mathcal{T}$ and the four root-to-leaf paths of a rectangle is shown. A rectangle will be stored (in a rank-space reduced manner) at the node of Stage-II and all the nodes on the four paths of Stage-III.

3.1 Our solution

Skeleton of the structure. Recall that the input is a set S of n 3-d 5-sided rectangles which are unbounded on one side along z-axis. Consider the projection of the rectangles of S onto the xy-plane and impose an orthogonal $\left[2\sqrt{\frac{n}{\log^4 n}}\right] \times \left[2\sqrt{\frac{n}{\log^4 n}}\right]$ grid such that each *horizontal* and *vertical slab* contains the projections of $\sqrt{n\log^4 n}$ sides of S. A horizontal (resp., vertical) slab is the region in the xy-plane which lies between two consecutive horizontal (resp., vertical) lines on the grid. This grid is associated to the root node of our tree $\mathcal{T}$. For each vertical and horizontal slab, we recurse on the rectangles of S which are *sent* to that slab. Below we will explain which rectangles are sent to which slab. At each node of the recursion tree, if we have m rectangles in the subproblem, the grid changes to $\left[2\sqrt{\frac{m}{\log^4 m}}\right] \times \left[2\sqrt{\frac{m}{\log^4 m}}\right]$. We stop the recursion when a node has less than $w^{1/4}$ rectangles.

Breaking the rectangles. The solution of Rahul [36] *breaks* each 5-sided rectangle only along x-axis obtain 4-sided rectangles, and then uses the solution for 4-sided rectangle stabbing as a black box. Unlike the approach of Rahul [36], we will break each 5-sided rectangle simultaneously along x- and y-axis to obtain $O(\log\log n)$ 3-sided rectangles.

For a node in the tree, the intersection of every pair of horizontal and vertical grid line defines a *grid point*. A rectangle $r \in S$ is associated with four root-to-leaf paths (as shown in Figure 2), which are described below. The four paths correspond to the four sides of the 5-sided rectangle when projected onto the xy-plane. Any node (say, v) on these four paths is classified w.r.t. r into one of the three stages as follows:

Stage-I. The xy-projection of r contains none of the grid points. Then r is not stored at v, and sent to the child corresponding to the row or column r lies in. If r lies inside a row and

a column, then it can be sent to either of them.

Stage-II. The xy-projection of r contains at least one of the grid points. Then r is broken into at most five disjoint pieces. The first piece is a *grid rectangle*, which is the bounding box of all the grid points lying inside r, as shown in Figure 3(b). The remaining four pieces are two *column rectangles* and two *row rectangles* as shown in Figure 3(c) and (d), respectively. The grid rectangle is stored at v. Note that each column rectangle (resp., row rectangle) is now a 3-d 4-sided rectangle w.r.t. its column (resp., row), and is sent to its corresponding child node.

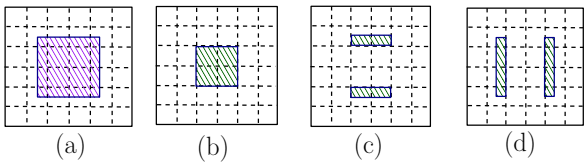


Figure 3: Breaking a 5-sided rectangle in (a) into a grid rectangle (as shown in (b)), two row rectangles (shown in (c)) and two column rectangles (shown in (d)).

Stage-III. The xy-projection of a 4-sided piece of r contains at least one of the grid points. Without loss of generality, assume that the 4-sided rectangle r is unbounded along the negative x-axis. Then the rectangle is broken into at most four disjoint pieces: a *grid rectangle*, two *row rectangles*, and a *column rectangle*, as shown in Figure 4(b), (c) and (d), respectively. The grid rectangle and the two row rectangles are stored at v, and the column rectangle is sent to its corresponding child node. Note that the two row rectangles are now 3-d 3-sided rectangles w.r.t. their corresponding rows (unbounded in one direction along x-, y- and z-axis).

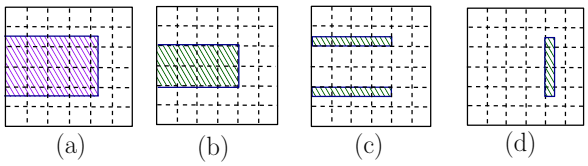


Figure 4: Breaking a 4-sided rectangle in (a) into a grid rectangle (as shown in (b)), two row rectangles (shown in (c)) and one column rectangle (shown in (d)).

Encoding structures. Let S_v be the set of rectangles stored at a node v in the tree. We apply a rank space reduction (mapping input coordinates to their ranks; rank-i coordinate is mapped to $2i$ on the integer grid) so that the coordinates of all the endpoints are in $[4|S_v|]^3$. If v is a leaf node, then we build an instance of Lemma 5. Otherwise, the following four structures will be built using S_v:

(A) Slow structure. An instance of Lemma 4 is built on S_v to answer the 3-d 5-sided rectangle stabbing query when the output size is "large".

(B) Grid structure. The intersection of a horizontal and a vertical slab is defined as a *cell*.

Therefore, an $[\alpha] \times [\beta]$ grid will have $\Theta(\alpha\beta)$ cells. For each cell c of the grid, among the rectangles which completely cover c, pick the $\log^3 |S_v|$ rectangles with the largest span along the z-axis. Store them in a list $Top(c)$ in decreasing order of their span. The span of a rectangle is its length along the z-axis.

(C) 3-d dominance structure. For a given row or column in the grid, only for the 3-sided rectangles stored in it, a linear-space 3-d dominance reporting structure [1, 12] is built. This structure is built for each row and column slab.

(D) Predecessor search structure. As mentioned above, the coordinates of all the endpoints of the rectangles are in $[4|S_v|]^3$. Therefore, to map the query point to $[4|S_v|]^3$, we will build three predecessor search structures [35] (one for each dimension) based on the endpoints obtained from the projection of the rectangles of S_v onto the x-, y-, and z-axis, respectively. The data structure occupies linear space and answers a predecessor query in $O(\log \log_w |S_v|)$ time.

Where are the original coordinates stored? Unlike the previous approaches for indexing points [7, 13], we use a somewhat unusual approach for storing the original coordinates of each rectangle. In the process of breaking each 5-sided rectangle described above, there will be four leaf nodes where portions of the rectangle will get stored. We will choose these leaf nodes to store the original coordinates of the rectangle (see Figure 2). The benefit is that each (rank-space reduced) rectangle stored at a node v has to maintain a decoding pointer of length merely $O(\log |S_v|)$ bits to point to its original coordinates stored in its subtree.

Query algorithm and analysis. Given a query point q, we start at the root node v and perform the following steps: Firstly, perform a rank space reduction of the query point to $[4|S_v|]^3$ using the predecessor search structures. Secondly, query the two dominance structures corresponding to the horizontal and the vertical slab containing q. Thirdly, for the grid structure, locate the cell c on the grid containing q. Scan the list $Top(c)$ to keep reporting till (a) all the rectangles have been exhausted, or (b) a rectangle not containing q is found. If case (a) happens and $|Top(c)| = \log^3 |S_v|$, then we discard the rectangles reported till now, and query the slow structure. The decoding pointers will be used to report the original coordinates of the rectangles. Finally, we recurse on the horizontal and the vertical slab containing q. If we visit a leaf node, then we query the leaf structure (Lemma 5).

First, we analyze the space. Let $s(|S_v|)$ be the *amortized* number of bits needed per input 5-sided rectangle in the subtree of a node v. The amortized number of bits needed per rectangle for the encoding structures and the pointers to the original coordinates is $O(\log |S_v|)$. This leads to the following recurrence:

$$s(n) = s(\sqrt{n \log^4 n}) + O(\log n)$$

which solves to $s(n) = O(\log n)$. Therefore, the overall space is bounded by $O(n)$ words.

Next, we analyze the query time. To simplify the analysis, we will exclude the output size term while computing the query time. At the root, the time taken to query the grid and the dominance structure is $O(\log \log_w n)$. This is followed by two recursive

calls corresponding to the horizontal and the vertical slab containing q. This leads to the following recurrence:

$$Q(n) = 2Q(\sqrt{n \log^4 n}) + O(\log \log_w n)$$

with a base case of $Q(w^{1/4}) = O(1)$. Via the substitution method, this solves to $Q(n) \leq c \cdot (\log_w n - \log \log_w n) = O(\log_w n)$, where c is a constant (see the appendix for the calculations). For each reported rectangle it takes constant time to recover its original coordinates. The time taken to query the slow structure is dominated by the output size. Therefore, the overall query time is $O(\log_w n + k)$.

Theorem 3. *There is a data structure of size $O(n)$ words that answers 3-d 5-sided rectangle stabbing queries in $O(\log_w n + k)$ time. From the lower bound in [34], it follows that our structure is optimal in the word RAM model.*

4 3-d 6-sided rectangle stabbing

In this section we will present the solution for 3-d 6-sided rectangle stabbing queries. The key technical contribution of this section is Lemma 7, which deals with answering a restricted version of "colored" 2-d dominance reporting query. The model of computation is the word RAM model.

4.1 Skeleton structure

Data structure. We construct an interval tree, IT, with fan-out $f = w^{\varepsilon}$ on the z-projections of rectangles in S for a positive constant $\varepsilon = 0.1$. For a node $v \in IT$, let $v_1, \ldots, v_f$ be its f children from left-to-right, and let $b_1(v), \ldots, b_{f+1}(v)$ be the $f+1$ bounding planes of its children, such that the slab of node v_i is bounded on the left (resp., right) by the plane b_i (resp., b_{i+1}), $\forall i \in [1, f]$. See Figure 5. The z-coordinate of a plane $b_i(v)$ is denoted by $z_i(v)$.

The set $S(v) \subseteq S$ contains all rectangles $[x_1, x_2] \times [y_1, y_2] \times [z_1, z_2]$, such that node v is the lowest common ancestor of the leaves corresponding to z_1 and z_2. We store three rectangle stabbing data structures $M(v)$, $L(v)$, and $R(v)$ at each node v. Consider an arbitrary rectangle $r = [x_1, x_2] \times [y_1, y_2] \times [z_1, z_2]$ stored in $S(v)$. Suppose that $z_k(v) < z_1 \leq z_{k+1}(v)$ and $z_{l-1}(v) \leq z_2 < z_l(v)$. If $k+1 < l-1$, we store a rectangle $[x_1, x_2] \times [y_1, y_2] \times [k+1, l-1]$ in the data structure $M(v)$. We also store r in data structures $R(v_k)$ and $L(v_l)$. The z-coordinates of all rectangles in $M(v)$ lie in the integer universe $[w^{\varepsilon}]$. We will use this fact to answer rectangle stabbing queries in $M(v)$ in $O(\log_w n + k)$ time, which will be shown later in this section in Theorem 5. Rectangles stored in $R(v)$ cross the rightmost bounding plane, $b_{f+1}(v)$, of node v. Similarly, rectangles stored in $L(v)$ cross the leftmost bounding plane, $b_1(v)$, of node v. Therefore, with respect to node v, we can treat the rectangles in $R(v)$ (resp. $L(v)$) as 5-sided rectangles. Using Theorem 3, we can answer stabbing queries on $R(v)$ and $L(v)$ in $O(\log_w n + k)$ time. The space occupied by the data structure will be $O(n)$, since each rectangle is stored at exactly one node in IT and the data structures of Theorem 5 and Theorem 3 occupy linear space.

Query algorithm. To report all rectangles that are stabbed by q, we traverse the path in IT from the root to the leaf that corresponds to the z-coordinate of q, and answer rectangle stabbing queries using data structures built for $L(v)$, $R(v)$ and $M(v)$ in each node v. Since the length of a root-to-leaf path is $O(\log_w n)$, the query is answered in $O(\log_w^2 n + k)$ time.

Theorem 4. *In the word RAM model, there is a linear-space data structure that answers 3-d 6-sided rectangle stabbing queries in $O(\log_w^2 n + k)$ time.*

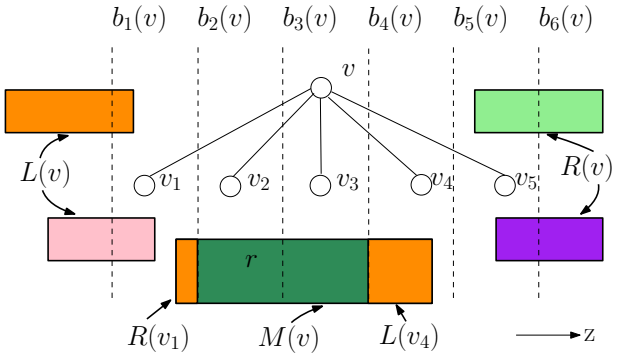


Figure 5: A node v in the interval tree with fanout five. A rectangle r in $S(v)$ is broken into three pieces and stored in three different structure $R(v_1), M(v)$ and $L(v_4)$.

4.2 z-restricted queries

A z-restricted rectangle in 3-d is of the form $[x_1, x_2] \times [y_1, y_2] \times [i, j]$, where integers $i, j \in [w^\varepsilon]$ and $\varepsilon = 0.1$. Rectangle stabbing queries on such rectangles will be called *z-restricted queries* and will be used to handle rectangles stored in $M(v)$. First, we will handle z-restricted 4-sided rectangles in 3-d of the form $r = (-\infty, x] \times (-\infty, y] \times [i, j]$.

Lemma 6. *There exists a linear-space data structure that answers z-restricted 3-d 4-sided rectangle stabbing queries in $O(\log w \cdot \log\log n + k)$ time.*

Proof. If the universe size is U along z-axis, then the structure of Rahul [36] leads to a query time of $O(\log U \cdot \log\log n + k)$. In our case $U = w^\varepsilon$. □

Now we will use the above lemma to build an optimal solution for z-restrictred 3-d 4-sided rectangle stabbing queries.

Lemma 7. *There exists a linear-space data structure that answers z-restricted 3-d 4-sided rectangle stabbing queries in $O(\log\log_w n + k)$ time.*

Proof. We can safely assume that $n > w^{2\varepsilon} \cdot \log w \log\log n$, because in the case of $n < w^{2\varepsilon} \cdot \log w \log\log n = O(w^{1/4})$, there is a linear-space data structure with query time $O(1+k)$ time, namely the structure of Lemma 5 (by replacing $\log n$ with w, the proof of Lemma 5 still holds).

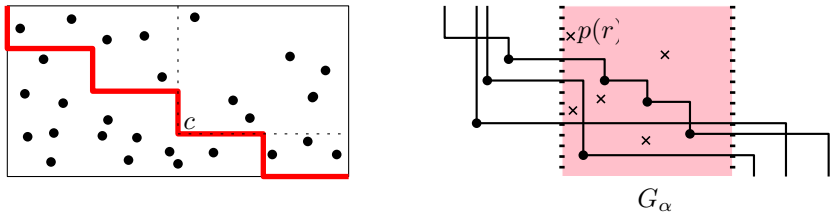
Shallow cuttings: Let P be a set of n points in 2-d. A point p_1 is said to *dominate* point p_2 if it has a larger x-coordinate and a larger y-coordinate value than those of p_2. Our main tool to handle the small values of k is *shallow cuttings for 2-d dominance ranges* [5] (see Figure 6(a)) which has the following three properties: (a) A t-shallow cutting for set P is a union of $O(n/t)$ cells where every cell is of the form $[a, +\infty) \times [b, +\infty)$, (b) any point in the 2-d plane that is dominated by at most t points of P will lie inside at least one cell, and (c) each cell contains at most $O(t)$ points of P. A cell $[a, +\infty) \times [b, +\infty)$ can be identified by its *corner* (a, b). We denote by $Dom(c)$ the set of points that dominate the corner c. For details on construction of shallow cuttings we refer the reader to Afshani *et al.* [5].

Data structure: We classify the z-restricted 4-sided rectangles according to their z-projections. The set S_{ij} contains all rectangles of the form $r = (-\infty, x_r] \times (-\infty, y_r] \times [i, j]$. Since $1 \leq i \leq j \leq w^{\varepsilon}$, there are $O(w^{2\varepsilon})$ sets S_{ij}. Every rectangle r in S_{ij} is associated with the point $p(r) = (x_r, y_r)$. A rectangle $r = (-\infty, x_r] \times (-\infty, y_r] \times [i, j]$ is stabbed by a query point $q = (q_x, q_y, q_z)$ if and only if $q_z \in [i, j]$ and the point $p(r)$ dominates the 2-d point (q_x, q_y). With this observation in mind, if $q_z \in [i, j]$, then computing $S_{ij} \cap q$ reduces to a 2-d dominance reporting query on $\bigcup_{r \in S_{ij}} p(r)$ with query point (q_x, q_y). Think of each pair (i, j) as a *color*. To achieve optimal query time, the challenge is to answer 2-d dominance query simultaneously on all colors, i.e., on $\bigcup_{r \in S_{ij}} p(r)$, where $i, j \in [w^{\varepsilon}]$.

Our first step is to construct a t-shallow cutting, say L_{ij}, with $t = \log w \cdot \log \log n$ for the set of points $p(r)$, such that $r \in S_{ij}$. This step is performed for each color. Next, we *group* corners of different shallow cuttings into one structure. Let $\mathcal{C}_{ij}$ denote the set of corners in a shallow cutting L_{ij} and let $\mathcal{C} = \bigcup_{i,j \in [w^{\varepsilon}]} \mathcal{C}_{ij}$. The set $\mathcal{C}$ is divided into disjoint groups, so that every group G_α consists of $w^{2\varepsilon}$ consecutive corners with respect to their x-coordinates. We say that a corner $c = (c_x, c_y)$ in $\mathcal{C}_{ij}$ is immediately to the left of G_α if it is the rightmost corner in $\mathcal{C}_{ij}$ such that $c_x \leq c'_x$ for any corner $c' = (c'_x, c'_y)$ in G_α. Now come the crucial definitions of $\overline{G}_\alpha$ and R_α. The set of corners $\overline{G}_\alpha$ contains (1) all corners from G_α, and (2) for every pair i, j such that $1 \leq i \leq j \leq w^{\varepsilon}$, the corner $c \in \mathcal{C}_{ij}$ immediately to the left of G_α. The set R_α contains all rectangles r such that $p(r) \in Dom(c)$ for at least one corner $c \in \overline{G}_\alpha$. See Figure 6(b). Since R_α contains $O(w^{2\varepsilon} \log w \cdot \log \log n) = O(w^{1/4})$ rectangles, we can perform a rank-space reduction and answer rectangle stabbing queries on R_α in $O(1 + k)$ time by using Lemma 5.

Next, we will show that the space occupied by this structure is $O(n)$. The crucial observation is that the number of corners in G_α is $w^{2\varepsilon}$ and the number of "immediately left" corners added to each $\overline{G}_\alpha$ is also bounded by $w^{2\varepsilon}$. The number of corners in set $\mathcal{C}$ is bounded by $\sum_{i,j} O\left(1 + \frac{|S_{ij}|}{t}\right) = O(w^{2\varepsilon} + n/t) = O(n/t)$, since $n/t > w^{2\varepsilon}$. Therefore, the number of groups will be $O\left(\frac{n}{tw^{2\varepsilon}}\right)$. Each set R_α contains $O(w^{2\varepsilon} \cdot t)$ rectangles. Therefore, the total space occupied by this structure is $\sum_\alpha |R_\alpha| = O\left(\frac{n}{tw^{2\varepsilon}} \cdot w^{2\varepsilon} \cdot t\right) = O(n)$.

Query algorithm: Given a query point $q = (q_x, q_y, q_z)$, we find the set G_α that "contains" q_x by a predecessor search. Then we report all the rectangles in R_α that are stabbed by q by using Lemma 5. If $|S_{ij} \cap q| < \log w \cdot \log \log n$ for all colors (i, j), then we claim that we can stop the query algorithm: fix any S_{ij} and consider any rectangle $r \in S_{ij}$. Since $|S_{ij} \cap q| < \log w \cdot \log \log n$, the number of points $p(r')$ such that $r' \in S_{ij}$ dominating r will



(a) (b)

Figure 6: (a) A 3-shallow cutting. Each cell here contains no more than six points (one such cell c is shown in the figure). Notice that any point in the plane which lies below the "staircase" is dominated by at least three black points. (b) The set of corners $\overline{G}_\alpha$ are represented as disks and the points corresponding to R_α are represented as crosses.

be less than $\log w \cdot \log\log n$. Therefore, by property (b) of shallow cuttings, there will be at least one cell in the $(\log w \cdot \log\log n)$-level shallow cutting L_{ij} which will contain the point $p(r)$. Our construction of $\overline{G}_\alpha$ ensures that the corner of at least one cell containing $p(r)$ is included in it. This ensures that rectangle r in indeed reported. Otherwise, if $|S_{ij} \cap q| \geq \log w \cdot \log\log n$ for at least one color (i, j), then we discard the set of rectangles reported till now, and instead query the structure of Lemma 6 to answer the query.

We need $O(\log\log_w n)$ time to find the group G_α [35], then perform a rank space reduction with respect to R_α in $O(\log_w |R_\alpha|) = O(1)$ time using a fusion tree [23], and then finally spend $O(1 + k)$ time to report $R_\alpha \cap q$. On the other hand, if we query the structure of Lemma 6, then the query time will be $O(\log w \cdot \log\log n + k) = O(k)$, since $k \geq \max\{|S_{ij} \cap q|\} \geq \log w \cdot \log\log n\}$. □

Now we turn our attention to z-restricted 6-sided rectangle stabbing queries. In this case, the data structure contains 6-sided rectangles, but again the z-coordinates of the endpoints lie in the integer universe $[w^\varepsilon]$.

Lemma 8. *There exists a linear-space data structure that answers z-restricted 6-sided rectangle stabbing queries in $O(\log n + k)$ time.*

Proof. By Theorem 3, a 5-sided rectangle stabbing query can be answered in $O(\log_w n + k)$ time. Plugging this result as a blackbox into the structure of Rahul [36] leads to a query time of $O(\log U \cdot \log_w n + k)$ for the 6-sided rectangle stabbing query, where U is the universe size along z-axis. In our case $U = w^\varepsilon$. □

Theorem 5. *There exists a linear-space data structure that answers a z-restricted 6-sided rectangle stabbing query in $O(\log_w n + k)$ time. This is optimal in the word RAM model.*

Proof. We use the same approach as in Theorem 3, but use different encoding structures.
(A) *Slow structure.* An instance of Lemma 8 is built on S_v.
(B) *Grid structure.* For every cell c we keep lists $Cover(c, z)$ for $z = 1, \ldots, w^\varepsilon$; $Cover(c, z)$ contains $\log |S_v|$ rectangles r, such that the projection of r onto the xy-plane completely covers c and the z-projection of r is stabbed by z.

(C) *"z-restricted dominance" structure.* For a given row or column in the grid, based on the z-restricted 4-sided rectangles stored in it, an instance of Lemma 7 is built.

The space and the query time analysis follow from the analysis of Theorem 3. □

Higher dimensions. By using segment trees [17], the result for 3-d rectangle stabbing can be extended to higher dimensions by paying a factor of $\log n$ per dimension for both space and query time: start by projecting all the n d-dimensional rectangles onto the first dimension and build a segment tree [17] based on the resulting n segments. Based on the rectangles (segments) corresponding to each node in the segment tree and the remaining $(d-1)$ dimensions, recursively build a $(d-1)$-dimensional rectangle stabbing structure.

Theorem 6. *There is a data structure of size $O(n \log^{d-3} n)$ that answers rectangle stabbing queries in d-dimensional space ($d \geq 4$) in $O(\log_w^2 n \cdot \log^{d-3} n + k)$ time.*

5 Top-k 2-d rectangle stabbing

Our solution for 5-sided rectangle stabbing can be modified to obtain an optimal solution for top-k 2-d stabbing query in the word RAM model (Theorem 8). Recall that in this problem we preprocess a set of weighted axis-aligned rectangles in 2-d, so that given a query point q and an integer k, the goal is to report the k largest-weight rectangles containing (or stabbed by) q. We use the same general approach, but we need additional ideas to handle the *top-k* aspect of the problem.

5.1 Top-k 2-d dominance queries

First, we will present an optimal solution for the top-k 2-d dominance queries where the input is a set P of n weighted points in 2-d, and the query is defined by an integer k and a dominance range $q = [q_x, \infty) \times [q_y, \infty)$. The result obtained is the following.

Theorem 7. *There exists a linear-space data structure that answers top-k 2-d dominance queries in $O(\log \log_w n + k)$ time. This is optimal in the word RAM model.*

The optimality of Theorem 7 follows from the lower bound of Patrascu and Thorup for the predecessor search problem [35]: let $p_1, p_2, \ldots, p_n$ be the coordinate values of the n points on $[U]$ in non-decreasing order, where U is a positive integer. Then a predecessor query on these points can be answered via a 1-d rectangle (interval) stabbing query on the intervals $[p_1, p_2), [p_2, p_3), \ldots, [p_n, \infty)$.

Preliminaries. We will need the following two building blocks for our solution.

Lemma 9. *(Patil* et al. *[33], Theorem 9.) There exists a linear-space data structure that answers top-k 2-d dominance queries in $O(\log n + k)$ time. The points are reported in sorted order of their weights.*

Lemma 10. *(Structure for few points.) Consider a set of $m = \log^{1/4} n$ points lying on an $[m] \times [m]$ grid with weights that are integers in the range $[1, m]$. Then there exists a data structure of size $O(m)$ words that answers top-k 2-d dominance queries in $O(1 + k)$ time. Here n is the number of points in the original top-k 2-d dominance query.*

Proof. The proof is similar to the proof of Lemma 5. The number of combinatorially different query regions will be $O(m^2)$ and the number of distinct values of k will be m. Therefore, the number of combinatorially different queries will be $O(m^3)$. Now following the proof of Lemma 5, the space can be bounded by $O(m)$ words and the query time by $O(1 + k)$. □

Shallow cuttings. We will re-define shallow cuttings for dominance ranges [5] in the context of 3-d points. A point p_1 is said to *dominate* point p_2 if it has a larger coordinate value in all the three dimensions. A t-shallow cutting for a set P of 3-d points is a collection of *boxes* of the form $[a, +\infty) \times [b, +\infty) \times [c, +\infty)$, such that (a) there are only $O(n/t)$ boxes, (b) every point that is dominated by at most t points from P will lie within some box, and (c) each box contains at most $O(t)$ points of P.

Consider a t-shallow cutting for a 3-d point set P. The z-range of a box $[a, +\infty) \times [b, \infty) \times [c, \infty)$ is defined as $[c, \infty)$. Given a 3-d point q, the FIND query either reports a box with the largest z-range in the shallow cutting which contains q or reports that there is no box which contains q. We will need the following fact about shallow cuttings.

Lemma 11. *(FIND query [1, 12].) There is a data structure of size $O(|P|)$ words which can answer the FIND query in $O(\log \log_w |P|)$ time.*

Data structure. Our structure consists of the following components:
(A) *Slow structure.* Based on the point set P, we build the data structure of Lemma 9.
(B) *$(\log n)$-level shallow cutting.* We regard the weight of the points of P as the third coordinate and construct a $(\log n)$-shallow cutting $\mathcal{P}_1$.
(C) *Slow structure for each box.* The *conflict list*, CL_B, of a box $B \in \mathcal{P}_1$ is the set of points in P which lie inside it. For each CL_B the data structure of Lemma 9 is constructed.
(D) *$(\log^{1/4} n)$-level shallow cutting.* A $(\log^{1/4} n)$-shallow cutting $\mathcal{P}_2$ is constructed based on the points in P.
(E) *Small-sized structures.* For every box in $\mathcal{P}_2$, based on its conflict list build the structure of Lemma 10.

The properties of the shallow cuttings and the fact that Lemma 9 and Lemma 10 are linear-space structures ensures that the space occupied by our data structure is $O(n)$ words.

Query algorithm. We will assume that $|P \cap q| \geq k$. (This is easy to check by reporting the points in $P \cap q$ till one of the following happens: either $k + 1$ points are reported or all the points are reported.) Define $q' = (q_x, q_y, +\infty)$. The query algorithm performs the following three steps:
(A) Perform the FIND operation on the $(\log^{1/4} n)$-shallow cutting $\mathcal{P}_2$ to find a box B which

contains q', perform a rank-space reduction of the query w.r.t. to CL_B, then query the small-sized structure built on CL_B, and stop the query algorithm. If no box contains q', then by property (b) of shallow cuttings, we conclude that $k = \Omega(\log^{1/4} n)$, and go to next step.
(B) Perform the FIND operation on the $(\log n)$-shallow cutting $\mathcal{P}_1$ to find a box B which contains q', then query the slow structure built on CL_B, and stop the query algorithm. If no box contains q', then we conclude that $k = \Omega(\log n)$, and go to next step.
(C) Query the slow structure.

Reporting at most $k+1$ points in $P \cap q$ takes $O(\log\log_w n + k)$ time [1, 12]. Step (A) takes $O(\log\log_w n + k)$ time, since the rank-space reduction can be done in $O(\log_w(\log^{1/4} n)) = O(1)$ time using a fusion-tree [23]. Step (B) takes $O(\log\log_w n + \log\log n + k) = O(k)$ time. Step (C) takes $O(\log n + k) = O(k)$ time. The overall query time is, therefore, $O(\log\log_w n + k)$.

5.2 Final solution for top-k 2-d rectangle stabbing

Now we are ready to prove the following result.

Theorem 8. *There is a linear-space data structure that can answer top-k 2-d rectangle stabbing queries in $O(\log_w n + k)$ time. This is optimal in the word RAM model.*

Preliminaries. As in the case of top-k 2-d dominance queries, we will need a slow structure and a small-sized structure.

Lemma 12. *There is a linear-space data structure that can answer top-k 2-d rectangle stabbing queries in $O(\log^2 n \cdot \log\log_w n + k)$ time.*

Proof. It is known that we can answer a 3-d 5-sided rectangle stabbing query by querying $O(\log^2 n)$ 3-d dominance reporting structures [36]. The space of the data structure is within the same bounds as the 3-d dominance reporting structure. Refer to [36] for further details.

In the same way, we can answer a 2-d top-k rectangle stabbing query by querying $O(\log^2 n)$ top-k 2-d dominance queries. We will return the k heaviest points in *sorted* order in $O(k)$ time using a technique that will be used later in this section. □

Lemma 13. *Consider a set of $m = w^{1/4}$ rectangles whose endpoints lie on an $[m] \times [m]$ grid, and have weights that are integers in the range $[1, m]$. Then there is a data structure of size $O(m)$ words that answers the top-k 2-d rectangle stabbing query in $O(1+k)$ time.*

All the above data structures (Theorem 7, Lemma 12, and Lemma 13) support queries in an online manner. That is, we do not need to know the value of k when the query is asked, since the elements are reported in descending order of their weights. The procedure can be paused and resumed at a later time.

Data structure. We only worry about $k < \log^3 n$, since the other case can be optimally handled by Lemma 12. We use the same structure as in the proof of Theorem 3 with the following differences: we use the top-k 2-d dominance structure of Theorem 7 instead of the 3-d dominance structure, and we use the slow data structure of Lemma 12 instead of the slow structure of Lemma 4. Now $Top(c)$ is the $\log^3 |S_v|$ heaviest rectangles among the rectangles covering the grid cell c and is stored in decreasing order of their weight. Finally, the leaf structure is built by using Lemma 13. Also, rank-space reduction on the points and the weights will be performed at each level of the recursion tree.

Query algorithm. Consider the following abstract problem: We are given sorted lists $L_1, L_2, \ldots, L_t$ such that the total number of elements in all the lists is less than or equal to n, and $t = O(\log_w n)$. The goal is to report the k heaviest elements. To answer this, we build a heap $\mathcal{G}$ based on the heaviest element from each list. Now we perform the following operations on $\mathcal{G}$ k times: (a) delete the element, e, with the heaviest weight and report it, and (b) if e came from list L_i, then insert the next heaviest element from L_i into $\mathcal{G}$. We implement $\mathcal{G}$ as a fusion tree [23]; since $\mathcal{G}$ contains at most $\log_w n$ elements, all operations on $\mathcal{G}$ are supported in $O(1)$ time. By now the reader must have guessed the query algorithm.
(A) *Identify* the $O(\log_w n)$ nodes as explained in Section 3: The height of the structure follows the recurrence $H(n) \leq H(\sqrt{n \log^4 n}) + 1$ with the base case of $H(w^{1/4}) = 1$, which solves to $H(n) = O(\log \log_w n)$, and for every node visited in the query algorithm in Section 3, two of its children are visited. As a result, the number of nodes visited (or identified) is $2^{H(n)} = O(\log_w n)$.
(B) Each identified node acts a list L_i (as discussed before, the grid structure and the top-k 2-d dominance structure can report in an online manner. If more than $\log^3 |S_v|$ rectangles have been reported from a node v, then we switch to its slow structure).
(C) Now find the top-k heaviest rectangles in $S \cap q$.

Note that the weights of the rectangles also undergo rank-space reduction. But for the abstract problem described above, the actual weight of the rectangles need to be known. This is easy to handle: each rectangle simply uses its decoding pointer to find its original coordinates and original weight which takes $O(1)$ time. This finishes the description of the algorithm.

6 Conclusion and Open Problems

In this paper we presented new results for orthogonal point location and rectangle stabbing. An intriguing question is to determine if our $O(\log_w n)$ query time bound for 3-d orthogonal point location for disjoint boxes is optimal, or if $(\log \log U)^{O(1)}$ bounds are at all possible, in the word RAM model. Also, we are not aware of any non-trivial lower bound for d-dimensional point location for disjoint boxes, to indicate that the number of logarithmic factors has to grow as d increases.

For 3-d 6-sided rectangle stabbing problem in the word RAM model, is our solution optimal? We conjecture that it is. On the upper-bound side, obtaining optimal solutions for 3-d 6-sided rectangle stabbing problem in the pointer machine model and the I/O model

is still open.

Finally, obtaining non-trivial upper bounds for 4-d rectangle stabbing problem in the word RAM model would be a challenging open problem. We refer the reader to [2, 3, 36] for the currently best known lower bounds and upper bounds for rectangle stabbing in higher dimensions in the pointer machine model.

References

[1] Peyman Afshani. On dominance reporting in 3D. In *Proceedings of European Symposium on Algorithms (ESA)*, pages 41–51, 2008.

[2] Peyman Afshani, Lars Arge, and Kasper Dalgaard Larsen. Orthogonal range reporting: query lower bounds, optimal structures in 3-d, and higher-dimensional improvements. In *Proceedings of Symposium on Computational Geometry (SoCG)*, pages 240–246, 2010.

[3] Peyman Afshani, Lars Arge, and Kasper Green Larsen. Higher-dimensional orthogonal range reporting and rectangle stabbing in the pointer machine model. In *Proceedings of Symposium on Computational Geometry (SoCG)*, pages 323–332, 2012.

[4] Peyman Afshani, Gerth Stølting Brodal, and Norbert Zeh. Ordered and unordered top-k range reporting in large data sets. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 390–400, 2011.

[5] Peyman Afshani and Konstantinos Tsakalidis. Optimal deterministic shallow cuttings for 3-d dominance ranges. *Algorithmica*, 80(11):3192–3206, 2018.

[6] Alok Aggarwal and Jeffrey Scott Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM (CACM)*, 31(9):1116–1127, 1988.

[7] Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. New data structures for orthogonal range searching. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 198–207, 2000.

[8] Lars Arge, Andrew Danner, and Sha-Mayn Teh. I/O-efficient point location using persistent B-trees. *ACM Journal of Experimental Algorithmics*, 8, 2003.

[9] J.L. Bentley. Solutions to Klee's rectangle problems. *Technical Report, Carnegie-Mellon University, Pittsburgh, PA*, 1977.

[10] Gerth Stølting Brodal. External memory three-sided range reporting and top-k queries with sublogarithmic updates. In *Proceedings of Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 23:1–23:14, 2016.

[11] Gerth Stølting Brodal, Rolf Fagerberg, Mark Greve, and Alejandro Lopez-Ortiz. Online sorted range reporting. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 173–182, 2009.

[12] 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.

[13] Timothy M. Chan, Kasper Green Larsen, and Mihai Pătraşcu. Orthogonal range searching on the RAM, revisited. In *Proceedings of Symposium on Computational Geometry (SoCG)*, pages 1–10, 2011.

[14] Timothy M. Chan and Gelin Zhou. Multidimensional range selection. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 83–92, 2015.

[15] Bernard Chazelle. Filtering search: A new approach to query-answering. *SIAM Journal of Computing*, 15(3):703–724, 1986.

[16] Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal of Computing*, 17(3):427–462, 1988.

[17] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3rd edition, 2008.

[18] Mark de Berg, Marc J. van Kreveld, and Jack Snoeyink. Two- and three-dimensional point location in rectangular subdivisions. *Journal of Algorithms*, 18(2):256–277, 1995.

[19] Herbert Edelsbrunner, Leonidas J. Guibas, and Jorge Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal of Computing*, 15(2):317–340, 1986.

[20] Herbert Edelsbrunner, G. Haring, and D. Hilbert. Rectangular point location in d dimensions with applications. *The Computer Journal*, 29(1):76–82, 1986.

[21] Herbert Edelsbrunner and Mark H. Overmars. On the equivalence of some rectangle problems. *Information Processing Letters (IPL)*, 14(3):124–127, 1982.

[22] Michael L. Fredman and Dan E. Willard. Blasting through the information theoretic barrier with fusion trees. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 1–7, 1990.

[23] Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences (JCSS)*, 47(3):424–436, 1993.

[24] Michael T. Goodrich, Mark W. Orletsky, and Kumar Ramaiyer. Methods for achieving fast query times in point location data structures. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 757–766, 1997.

[25] Michael T. Goodrich, Jyh-Jong Tsay, Darren Erik Vengroff, and Jeffrey Scott Vitter. External-memory computational geometry. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 714–723, 1993.

[26] Sathish Govindarajan, Pankaj K. Agarwal, and Lars Arge. CRB-tree: An efficient indexing scheme for range-aggregate queries. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 143–157, 2003.

[27] John Hershberger, Subhash Suri, and Csaba D. Tóth. Binary space partitions of orthogonal subdivisions. *SIAM Journal of Computing*, 34(6):1380–1397, 2005.

[28] John Iacono and Stefan Langerman. Dynamic point location in fat hyperrectangles with integer coordinates. In *Proceedings of the Canadian Conference on Computational Geometry (CCCG)*, 2000.

[29] Joseph JáJá, Christian Worm Mortensen, and Qingmin Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 558–568, 2004.

[30] David G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal of Computing*, 12(1):28–35, 1983.

[31] Richard J. Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM Journal of Computing*, 9(3):615–627, 1980.

[32] Yakov Nekrich. I/O-efficient point location in a set of rectangles. In *Latin American Symposium on Theoretical Informatics (LATIN)*, pages 687–698, 2008.

[33] Manish Patil, Sharma V. Thankachan, Rahul Shah, Yakov Nekrich, and Jeffrey Scott Vitter. Categorical range maxima queries. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 266–277, 2014.

[34] Mihai Pătraşcu. Unifying the landscape of cell-probe lower bounds. *SIAM Journal of Computing*, 40(3):827–847, 2011.

[35] Mihai Pătraşcu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 232–240, 2006.

[36] Saladi Rahul. Improved bounds for orthogonal point enclosure query and point location in orthogonal subdivisions in $\mathbb{R}^3$. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 200–211, 2015.

[37] Saladi Rahul and Ravi Janardan. A general technique for top-k geometric intersection query problems. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 26(12):2859–2871, 2014.

[38] Saladi Rahul and Yufei Tao. On top-k range reporting in 2d space. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 265–275, 2015.

[39] Saladi Rahul and Yufei Tao. Efficient top-k indexing via general reductions. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 277–288, 2016.

[40] Neil Sarnak and Robert Endre Tarjan. Planar point location using persistent search trees. *Communications of the ACM (CACM)*, 29(7):669–679, 1986.

[41] Cheng Sheng and Yufei Tao. Dynamic top-k range reporting in external memory. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 121–130, 2012.

[42] Qingmin Shi and Joseph JáJá. Novel transformation techniques using q-heaps with applications to computational geometry. *SIAM Journal of Computing*, 34(6):1474–1492, 2005.

[43] Jack Snoeyink. Point location. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 767–787. CRC Press, 2nd edition, 2004.

[44] Yufei Tao. A dynamic I/O-efficient structure for one-dimensional top-k range reporting. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 256–265, 2014.

Appendix: Solving the recurrence

In the following discussion, the variables $c, c_1, c_2, c_3,$ and C are constants independent of n. We are given that $Q(n) = 2Q(\sqrt{n \log^4 n}) + c_1 \cdot \log\log_w n$ with a base case of $Q(w^{1/4}) \leq c_2$. Expanding the above recurrence a constant number of times, it is easy to show that $Q(w^{256}) \leq f(c_1, c_2)$, where $f(c_1, c_2)$ is a constant which depends on the values of c_1 and c_2. Let $C = f(c_1, c_2)$.

We will prove that the above recurrence solves to $Q(n) \leq c \cdot (\log_w n - \log\log_w n) + C$. For $w^{1/4} \leq n \leq w^{256}$, a lower bound for the function $c \cdot (\log_w n - \log\log_w n) + C$ is $c \cdot (\log_w w^{1/4} - \log\log_w w^{1/4}) + C \geq C$. On the other hand, for $w^{1/4} \leq n \leq w^{256}$, an upper bound for the function $Q(n)$ is $Q(w^{256}) \leq C$. Therefore, $Q(n) \leq c \cdot (\log_w n - \log\log_w n) + C$, for $w^{1/4} \leq n \leq w^{256}$.

Next, we will prove $Q(n) \leq c \cdot (\log_w n - \log\log_w n) + C$ for all $n \geq w^{256}$ via the substitution method.

$$\begin{aligned}
Q(n) &\leq 2\left(c \cdot \left(\log_w \sqrt{n \log^4 n} - \log\log_w \sqrt{n \log^4 n}\right) + C\right) + c_1 \cdot \log\log_w n \\
&\leq c\log_w n + c\log_w \log^4 n - 2c\log\log_w \sqrt{n} - 2c\log\log_w \log^2 n + 2C + c_1 \cdot \log\log_w n \\
&\leq c\log_w n + c\log_w w^4 - 2c\log\log_w \sqrt{n} - 2c\log\log_w w^2 + 2C + c_1 \cdot \log\log_w n \\
&\leq c\log_w n + 4c - (2c\log\log_w n - 2c) - 2c + 2C + c_1 \cdot \log\log_w n \\
&\leq c\log_w n - 2c\log\log_w n + c_1 \cdot \log\log_w n + 4c + 2C \\
&\leq (c \cdot (\log_w n - \log\log_w n) + C) + (c_1 \log\log_w n - c\log\log_w n + 4c + C) \\
&\leq c \cdot (\log_w n - \log\log_w n) + C,
\end{aligned}$$

where the last inequality uses the fact that $c_1 \log\log_w n - c\log\log_w n + 4c + C \leq 0$, which is proven next. We choose c sufficiently large such that $c/2 \geq c_1 + C$. Then,

$$\begin{aligned}
& c_1 \log\log_w n - c\log\log_w n + 4c + C \\
\leq & (c_1 + C)\log\log_w n - c\log\log_w n + 4c, \text{ since } \log\log_w n \geq 1 \\
\leq & -\frac{c}{2}\log\log_w n + 4c \leq 0, \text{ since } n \geq w^{256}.
\end{aligned}$$