

# New Data Structures for Orthogonal Range Reporting and Range Minima Queries

Yakov Nekrich\*

## Abstract

In this paper we present new data structures for two extensively studied variants of the orthogonal range searching problem.

First, we describe a data structure that supports two-dimensional orthogonal range minima queries in  $O(n)$  space and  $O(\log^\varepsilon n)$  time, where  $n$  is the number of points in the data structure and  $\varepsilon$  is an arbitrarily small positive constant. Previously known linear-space solutions for this problem require  $O(\log^{1+\varepsilon} n)$  time (Chazelle, 1988) or  $O(\log n \log \log n)$  time (Farzan et al., 2012). A modification of our data structure uses space  $O(n \log \log n)$  and supports range minima queries in time  $O(\log \log n)$ . Both results can be extended to support three-dimensional five-sided reporting queries.

Next, we turn to the four-dimensional orthogonal range reporting problem and present a data structure that answers queries in optimal  $O(\log n / \log \log n + k)$  time, where  $k$  is the number of points in the answer. This is the first data structure that achieves the optimal query time for this problem.

Our results are obtained by exploiting the properties of three-dimensional shallow cuttings.

## 1 Introduction

Orthogonal range searching is a fundamental and extensively studied data structuring problem [20, 8, 13, 14, 12, 9, 29, 10, 11, 32, 33, 2, 3, 23, 22, 24, 19, 28, 6, 5]. In this problem we store a set of multi-dimensional points  $P$  in a data structure so that for an arbitrary axis-parallel rectangle  $Q$  some information about points in  $Q \cap P$  must be returned. Different variants of range searching queries have been studied by researchers: an orthogonal range reporting query asks for the list of all points in  $P \cap Q$ ; an orthogonal range emptiness query determines whether  $P \cap Q = \emptyset$ , an orthogonal range counting query asks for the number of points in  $P \cap Q$ . In the range minima/maxima problem each point is assigned a priority and we must return the point of smallest/highest priority in  $P \cap Q$ .

In this paper we study the orthogonal range reporting and range minima problems. We improve the query time of linear-space range minima data structure in two dimensions from  $O(\log n \log \log n)$  to  $O(\log^\varepsilon n)$ . Henceforth  $n$  denotes the total number of points in the data structure and  $\varepsilon$  is an arbitrarily small positive constant. We also describe a data structure with optimal  $O(\log n / \log \log n)$  query time for the four-dimensional orthogonal range reporting problem.

**Range Minima Queries.** The best previously known trade-offs are listed in Table 1. The study of compact data structures for range searching problems was initiated by Willard [34] and Chazelle [9]. In the latter work, published over three decades ago, Chazelle [9] described an  $O(n)$ -space data structure that supports two-dimensional range minimum queries<sup>1</sup> in  $O(\log^{1+\varepsilon} n)$  time. The only improvement for an  $O(n)$ -space data structure was achieved by Farzan et al. [15] who reduced the query time to  $O(\log n \log \log n)$ .

Better query times for this problem can be achieved at a cost of increasing the space usage. Chan et al. [6] described a data structure that uses  $O(n \log^\varepsilon n)$  space and answers queries in optimal  $O(\log \log n)$  time. Another trade-off was achieved by Karpinski and Nekrich[19]; combining their data structure with the result of [5], we can obtain a data structure that uses  $O(n(\log \log n)^3)$  space and answers queries in  $O((\log \log n)^2)$  time. A five-sided range reporting query is a special case of three-dimensional orthogonal range reporting queries where a query is bounded on five sides. Range minima problem is closely related to the special case of three-dimensional range reporting when the query range is bounded on five sides. All previous results, except for [15], can be extended to support five-sided queries.

Range minima queries are to be contrasted with two-dimensional emptiness queries. In this problem we store a set of two-dimensional points; given an axis-parallel query rectangle  $Q$ , we must decide whether  $Q \cap P = \emptyset$ . Emptiness queries can be answered in

\*Department of Computer Science, Michigan Technological University. Email [yakov.nekrich@googlemail.com](mailto:yakov.nekrich@googlemail.com)

<sup>1</sup>Range minima and range maxima problems are equivalent. In this paper we will talk about the range minima problem.

$O(\log^\varepsilon n)$  time using an  $O(n)$ -space data structure [6], or in  $O(\log \log n)$  time using an  $O(n \log \log n)$ -space data structure [2, 6].

In this paper we demonstrate that the gap between range emptiness and range minima in two dimensions can be closed completely. We present a data structure that uses  $O(n)$  space and answers range minima queries in  $O(\log^\varepsilon n)$  time. We also describe a data structure that uses  $O(n \log \log n)$  space and answers queries in  $O(\log \log n)$  time. Our results can be extended to five-sided queries in three dimensions. Another corollary of our result is a new trade-off for general (six-sided) range reporting in three dimensions.

Our data structure employs the standard recursive grid approach frequently used in orthogonal range searching problems [2, 6, 15, 19, 7]. The novel part of our method is a compact data structure supporting three-dimensional dominance queries for each recursive sub-structure of the grid. This data structure is based on the notion of a  $t$ -shallow cutting. We show that a shallow cutting can be "covered", in a certain sense, by a set of rectangles. Every covering rectangle contains a small number of points and is unbounded along the third dimension. We can specify the relevant points by their positions in rectangles. This approach essentially reduces the problem of storing points in a recursive grid to the problem of storing points in a compact range tree aka the ball inheritance problem [9, 6]. Another interesting feature of our approach is encoding of three-dimensional points as answers to two-dimensional queries, somewhat similar to the approach of Farzan et al. [15].

**Multi-Dimensional Range Reporting.** Orthogonal range reporting queries can be answered in  $O(\log \log n + k)$  time in two [29, 2] and three dimensions [5]. By the lower bound for predecessor queries [31], this query time is optimal. The best previously known four-dimensional data structure supports queries in  $O(\log n + k)$  time. According to the lower bound of Pătraşcu [30], any data structure that consumes  $O(n \text{polylog}(n))$  space requires  $\Omega(\log n / \log \log n)$  time to answer four-dimensional queries; this lower bound is also valid for emptiness queries. In this paper we describe, for the first time, a data structure that achieves the optimal  $O(\log n / \log \log n + k)$  query time for the four-dimensional range reporting problem. Henceforth  $k$  denotes the number of points in the query range.

Previous solutions of this problem employed range trees to solve the orthogonal range reporting problem in four dimensions. To answer a query, we must navigate

a node-to-leaf path<sup>2</sup> in a range tree and answer a three-dimensional range reporting query in every node on that path. By the lower bound for three-dimensional reporting, we have to spend  $\Omega(\log \log n)$  time in every visited node. The node degree of the range tree is bounded by  $\log^{O(1)} n$ ; a higher node degree would lead to prohibitively high space usage because we must store a separate data structure for every range of node children. Thus we must navigate along a path of  $\Omega(\log n / \log \log n)$  nodes and spend  $\Omega(\log \log n)$  time in every node. For this reason all previous methods need  $\Omega(\log n)$  time.

Again, in this paper we achieve better query time by using  $t$ -shallow cuttings. Our solution is based on embedding a high-degree tree  $T_0$  into the range tree  $T$  and storing  $t$ -shallow cuttings in the nodes of  $T_0$ . These  $t$ -shallow cuttings provide us with additional information and enable us to spend  $o(\log \log n)$  time in every visited node of  $T$  when a query is answered. In order to achieve the optimal query time, we use a sequence of embedded trees  $T_0, T_1, \dots$  with decreasing node degrees.

Throughout this paper  $n$  will denote the total number of points in a data structure. The number of points in a sub-structure of a global structure will be sometimes denoted by  $m$ . We assume w.l.o.g. that all point coordinates are positive integers bounded by  $n$  and all points have different coordinates. The general case can be reduced to this case by applying the reduction to rank space technique. Our results are valid in the standard RAM model. In this model we assume that the word size is  $\Theta(\log n)$  and that standard arithmetic operations can be performed on words in constant time. (In some cases our methods also make use of "non-standard" operations. However we can always implement these operations with table look-ups. The necessary look-up tables can be initialized in  $O(n^\varepsilon)$  time.). The space usage is measured in words of  $\log n$  bits, unless specified otherwise.

First we describe the linear-space data structure for five-sided three-dimensional range reporting. We explain how the standard grid approach in Section 2. We prove our result about "covering" a shallow cutting by rectangles in Section 3 and show how this covering can be used to obtain a compact dominance reporting data structure in Section 4. The same data structure can be easily modified to support two-dimensional range minima queries. Using the same approach, we obtain an  $O(n \log \log n)$ -space data structure for range minima and five-sided reporting queries with  $O(\log \log n)$  (resp.

<sup>2</sup>Depending on the type of the query and the data structure, we may have to navigate along  $O(1)$  different node-to-leaf paths. For simplicity, we discuss the case of exactly one path.

Ref.	Space	Range-Minima	Five-sided
[9]	$O(n)$	$O(\log^{1+\varepsilon} n)$	$O(\log^{1+\varepsilon} n + k \log^\varepsilon n)$
[15]	$O(n)$	$O(\log n \log \log n)$	-
[6]	$O(n \log^\varepsilon n)$	$O(\log \log n)$	$O(\log \log n + k)$
[19] + [5]	$O(n(\log \log n)^3)$	$O(\log \log n)^2$	$O((\log \log n)^2 + k \log \log n)$
Our	$O(n)$	$O(\log^\varepsilon n)$	$O((k+1) \log^\varepsilon n)$
Our	$O(n \log \log n)$	$O(\log \log n)$	$O((k+1) \log \log n)$

Table 1: Previous best and new results on range minima and five-sided range reporting problems.

$O((k+1) \log \log n)$  query time; this result is described in Section 5. The data structure for four-dimensional orthogonal range reporting is presented in Section 7.

## 2 Recursive Grid

We divide the  $(x, y)$  grid into  $\sqrt{n/\log^3 n}$  vertical slabs and  $\sqrt{n/\log^3 n}$  horizontal slabs, so that each slab contains  $\sqrt{n \log^3 n}$  points. For every slab we keep a data structure supporting three-dimensional dominance queries, that will be described in Section 4. This data structure uses  $O(\log \log n)$  bits per point and answers queries in  $O((k+1) \log^\varepsilon n)$  time. The top data structure  $D^t$  contains the  $\log^{1+\varepsilon} n$  points with smallest  $z$ -coordinates from every cell. Last, we store a recursively defined data structure for every slab that contains  $\Omega(\log^8 n)$  points. When the number of points in a slab does not exceed  $O(\log^8 n)$ , we keep all points in a data structure that uses  $O(\log \log n)$  bits per point and supports queries in  $O(1+k)$  time. This data structure can be constructed using standard techniques; see Section 8.

We observe that an  $O(n)$ -word data structure supporting five-sided queries in  $O(\log^{1+\varepsilon} n + k \log^\varepsilon n)$  time is already known [9]. In Sections 3 - 4 we describe the data structure for *log  $n$ -capped* queries: we report all  $k$  points in the query range if  $k \leq \log n$ ; if  $k > \log n$ , we return *NULL*. In the latter case, we can use the "slow" data structure of Chazelle and report all points in the query range in time  $O(\log^{1+\varepsilon} n + k \log^\varepsilon n) = O((1+k) \log^\varepsilon n)$ .

A four-sided query (i.e., a 3-d query bounded on four sides) can be answered as follows. If a query range  $[a, b] \times [0, h] \times [0, z]$  is entirely contained in one horizontal slab  $H$ , we answer the query using the data structure for  $H$ . If a query is contained in one vertical slab  $V$ , we answer the query using the data structure for  $V$ . Suppose the query intersects several horizontal slabs and several vertical slabs. Let  $H_j$  denote the horizontal slab that contains  $h$ ; let  $V_a$  and  $V_b$  denote vertical slabs that contains  $a$  and  $b$  respectively. A query is split into four parts, see Fig. 1. The central part is aligned

with slab boundaries, three other parts are contained in slabs  $V_a$ ,  $V_b$ , and  $H_j$  respectively. Let  $a'$  denote the  $x$ -coordinate of the right boundary of  $V_a$  and let  $b'$  denote the  $x$ -coordinate of the left boundary of  $V_b$ . Let  $h'$  denote the  $y$ -coordinate of the lower boundary of  $H_j$ . We can report points in  $[a', b'] \times [0, h']$  using the top data structure. We ask dominance queries  $[0, b] \times [0, h]$  and  $[a, +\infty) \times [0, h]$  to slabs  $V_a$  and  $V_b$  respectively. We ask a three-sided query  $[0, h] \times [a', b']$  to horizontal slab  $H_j$ . A query to the top data structure and two dominance queries take  $O(\log^\varepsilon n)$  time. The total query time is  $q(n) = O(\log^\varepsilon n) + q(\sqrt{n \log^3 n}) = O(\log^\varepsilon n \log \log n)$ .

A five-sided query  $[a, b] \times [h_1, h_2] \times [0, z]$  is processed in a similar way. If a query is entirely contained in one horizontal or vertical slab, we answer the query using the data structure for that slab. If a query intersects several horizontal slabs and several vertical slabs, we split the query range into five parts. Let  $V_a$  and  $V_b$  denote the vertical slabs that contain  $a$  and  $b$  respectively; let  $H_1$  and  $H_2$  denote the horizontal slabs that contain  $h_1$  and  $h_2$ . We answer three-sided queries  $[a, +\infty) \times [h_1, h_2]$  and  $[0, b] \times [h_1, h_2]$  on slabs  $V_a$  and  $V_b$  respectively. We answer two other three-sided queries  $[a, b] \times [h_1, +\infty)$  and  $[a, b] \times [0, h_2]$  on slabs  $H_1$  and  $H_2$ . The central part of the query can be answered using the top data structure  $D^t$ . The total query time is dominated by three-sided queries,  $q(n) = O(\log^\varepsilon n \log \log n)$ . We can reduce the time to  $O(\log^\varepsilon n)$  by replacing  $\varepsilon$  with an arbitrary  $\varepsilon' < \varepsilon$  in the above construction.

Let  $S(n)$  denote the space usage of our structure in bits. The top data structure can be implemented using e.g. [19] and requires  $O((n/\log^3 n) \log^{2+\varepsilon}) = o(n)$  bits. Dominance data structures use  $O(n \log \log n)$  bits because a dominance data structure consumes  $O(\log \log n)$  bits per point and each point is kept in two slabs. Hence  $S(n) = O(n \log \log n) + 2\sqrt{n/\log^3 n} S(\sqrt{n \log^3 n})$ . We set  $c(n) = S(n)/n$  and divide both parts of the previous equality by  $n$ . Thus  $c(n) = O(\log \log n) + 2c(\sqrt{n \log^3 n})$ . The latter recursion can be resolved to  $c(n) = O(\log n)$ . Hence  $S(n) = O(n \log n)$  and the data structure uses  $O(n)$  words of  $\log n$  bits.

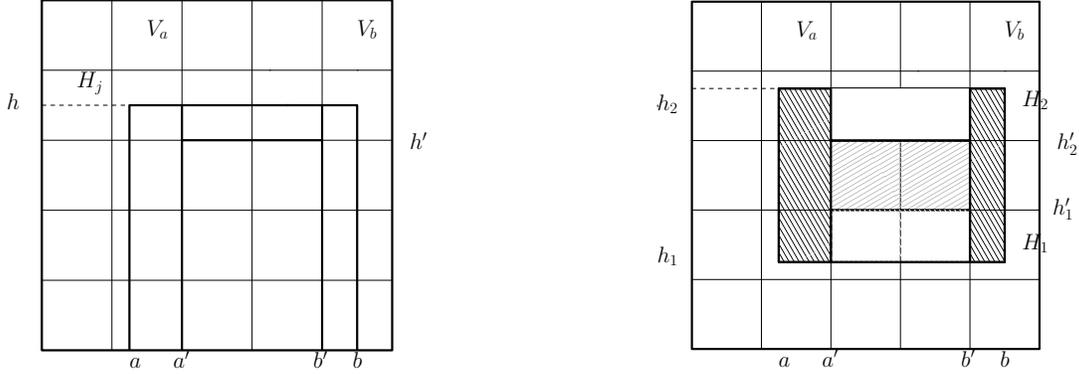


Figure 1: Left: decomposition of a four-sided query. Right: decomposition of a five-sided query. Only projections of slabs onto the  $(x, y)$ -plane are shown.

**THEOREM 2.1.** *There exists a data structure that uses  $O(n)$  words of space and supports five-sided queries in  $O((k+1)\log^\varepsilon n)$  time.*

### 3 Covering of a Shallow Cutting

Our main tool in designing a compact dominance data structure is three-dimensional shallow cuttings [1]. A  $t$ -shallow cutting for a set  $P$  is a collection of  $O(n/t)$  cells such that each cell  $C$  is a rectangle of the form  $[0, a] \times [0, b] \times [0, c]$ . Furthermore each cell contains at most  $2t$  points from  $P$  and every point  $q$  in 3-d space that dominates at most  $t$  points from  $P$  is contained in some cell(s). The list of all points from  $P$  in a cell  $C$  is called a conflict list of  $C$  and is denoted by  $list(C)$ . For a cell  $C = [0, a] \times [0, b] \times [0, c]$ , the point  $(a, b, c)$  is called the corner of  $C$ .

In this section we show that conflict lists of all cells in a  $t$ -shallow cutting can be almost covered by 3-d boxes unbounded in  $z$ -direction. The conflict list of each cell is contained in  $O(d^3)$  boxes where  $d$  is a parameter that does not depend on  $t$ . There can be a small number of points that is not contained in these  $O(d^3)$  boxes. However the total number of such points for all conflict lists is  $O(|P|/d)$ .

**THEOREM 3.1.** *Let  $\mathcal{C}$  denote a  $t$ -shallow cutting of a three-dimensional point set  $P$ ,  $|P| = m$ . For any integer  $d > 0$  there exists a subset  $P'$  of  $P$  and a set of three-dimensional rectangles  $\mathcal{R} = \{R_1, R_2, \dots, R_s\}$ , such that*

- (a)  $|P'| = O(m/d)$
- (b) Rectangles  $R_i$  are unbounded along the  $z$ -axis.
- (c) The conflict list of any cell, except for points from

$P'$ , is contained in  $g = O(d^3)$  rectangles from  $\mathcal{R}$ ,

$$list(C_i) \cap (P \setminus P') \subseteq (list(C_i) \cap R_{i_1}) \cup (list(C_i) \cap R_{i_2}) \cup \dots \cup (list(C_i) \cap R_{i_g})$$

- (d) Each rectangle contains  $O(t \cdot d^4)$  points of  $P$ .

*Proof.* We represent  $P'$  as a union of three sets  $P' = P_1 \cup P_2 \cup P_3$ .  $P_1$  is the subset containing all points from  $P$  that are stored in conflict lists of at least  $d$  different cells for a parameter  $d$ . The number of points in  $P_1$  is at most  $O(n/d)$ . Next we will construct the set  $P_2$  and the set of rectangles  $\mathcal{R}$ , so that conditions (b) and (c) are satisfied. Finally we will remove some rectangles from  $\mathcal{R}$  and construct  $P_3$ , so that condition (d) is satisfied. For simplicity we sometimes do not distinguish between a rectangle or a point and its projection on the  $xy$ -plane.

**Constructing  $P_2$ : Staircases, Regions, and Neighborhoods.** Consider the corners  $c_i$  of all cells  $C_i$  in a shallow cutting. We can assume that no  $c_i$  is dominated by another corner  $c_j$  (if this is the case,  $C_i$  is contained in  $C_j$  and we can remove the cell  $C_i$  from the shallow cutting).

Let  $M$  be the set of projections of corners onto the  $xy$ -plane. We decompose  $M$  into maximal layers (layers of maxima)  $M_i$ :  $M_1$  is the set of maximal points<sup>3</sup> of  $M$ .  $M_2$  is the set of maximal points of  $M \setminus M_1$  and  $M_i$  for  $i > 2$  is the set of maximal points in  $M \setminus (\cup_{j=1}^{i-1} M_j)$ . Thus every point on  $M_i$  is dominated by some point on  $M_{i-1}$  and no point on  $M_i$  is dominated by another point on  $M_i$ . We connect points of  $M_i$  by alternating horizontal and vertical segments; the resulting polyline will be called the *staircase* of  $M_i$ . See Fig. 2.

We visit corners of  $\mathcal{C}$  on a layer  $M_i$  in the decreasing order of their  $z$ -coordinates. Let  $c_j$  denote the  $j$ -th

<sup>3</sup>A point  $p$  in a set  $M$  is maximal if  $p$  is not dominated by any other point in  $M$ .

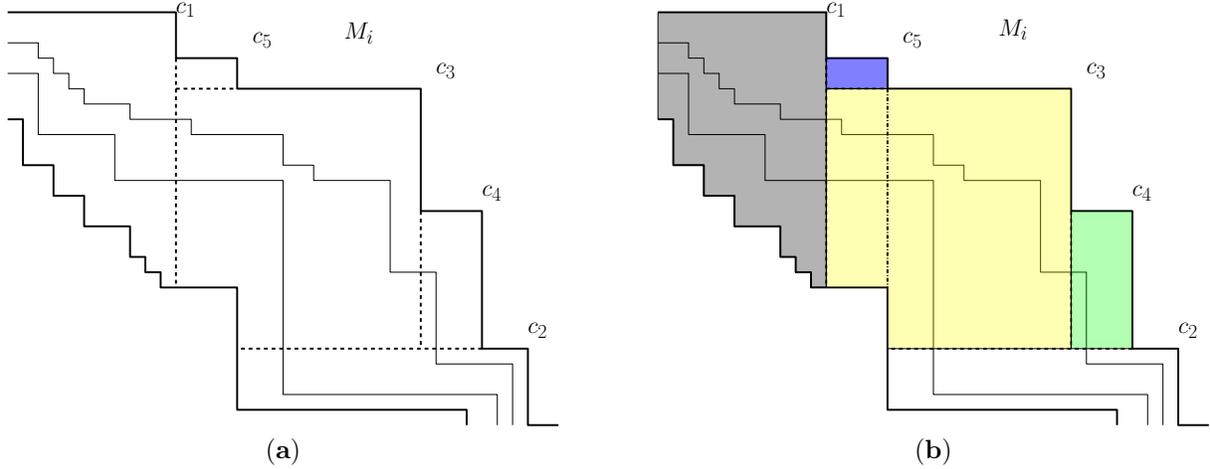


Figure 2: Division of a  $t$ -shallow cutting into regions. Left: shooting vertical and horizontal rays from corners of  $M_i$ . We assume that  $d = 3$  and  $z(c_i) < z(c_{i+1})$  for  $1 \leq i \leq 5$ . Right: regions associated to corners  $c_3, c_4, c_5$  and partially  $c_1$  are shown in different colors. The region of  $c_1$  is empty. The region of  $c_3$ , shown in yellow, is divided into two rectangles. The regions of  $c_5$  and  $c_4$  consist of one rectangle each.

visited corner on  $M_i$ . We shoot a horizontal ray in  $-x$  direction from  $\pi(c_j)$  until it hits either a ray of a previously visited corner  $c_l$ ,  $l < j$ , or the staircase of  $M_{i+d}$ . We also shoot a vertical ray from  $\pi(c_j)$  until it hits either a ray of a previously visited corner  $c_{l'}$ ,  $l' < j$ , or the staircase of  $M_{i+d}$ . We will call the polygon bounded by two rays from  $\pi(c_j)$ , the rays from  $c_l$  and  $c_{l'}$  and a portion of the staircase  $M_{i+d}$  the *region* of  $c_j$ .

If the region of  $c_j$  contains at least  $d^2$  corners of  $M_l$  for all  $l$  such that  $i < l \leq i + d$ , we add all points of  $\text{list}(C_j)$  to  $P_2$  and say that the region of  $c_j$  is empty. Otherwise we divide the region of  $c_j$  into at most  $O(d^2)$  rectangles, called rectangles of  $c_j$  (or rectangles associated to  $c_j$ ). See Fig. 2. The set of rectangles  $\mathcal{R}$  consists of all rectangles associated to corners of  $\mathcal{C}$ .

The *neighborhood* of a corner  $c_j$  is defined as follows. We shoot a horizontal ray in  $-x$  direction until we either (i) cross the boundaries of  $d$  non-empty regions or (ii) hit the staircase of  $M_{i+d}$ . We also shoot a vertical ray in  $-y$  direction until we either (i) cross the boundaries of  $d$  non-empty regions or (ii) hit the staircase of  $M_{i+d}$ . We call all rectangles whose boundaries are crossed by the vertical and the horizontal rays from  $c_j$  the *neighbors* of  $c_j$ .

LEMMA 3.1. *There are at most  $O(d^3)$  rectangles associated to neighbors of a corner  $c_j$ .*

*Proof.* Each corner has at most  $2d$  neighbor regions that are not empty. Every non-empty region is divided into  $O(d^2)$  rectangles.  $\square$

LEMMA 3.2. *For any point  $p$  in  $\text{list}(C_j)$ , either  $p \in P_1 \cup P_2$  or  $p$  is contained in some rectangle  $R_{j_l}$  associated to a neighbor of  $c_j$ .*

*Proof.* Consider a corner  $c_j$  and a point  $p \in \text{list}(C_j)$  that is not contained in any rectangle associated to a neighbor of  $c_j$ . The following cases are possible: (1)  $p$  is contained in an empty region of some neighbor  $c_l$  of  $c_j$ . By definition of a region,  $l < j$ , and  $z(c_l) \geq z(c_j)$ . Hence  $p \in \text{list}(C_l)$  and  $p \in P_2$ . (2)  $p$  is dominated by (corners of) at least  $d$  non-empty regions. In this case  $\pi(p)$  is dominated by at least  $d$  corners  $\pi(c_l)$  for some  $l < j$ . Since  $z(c_l) > z(c_j) > z(p)$ ,  $p$  is contained in the conflict list of every such  $C_l$  and  $p \in P_1$ . (3)  $p$  is dominated by some corner of  $M_{i+d}$ . In this case we can show that  $p$  is in at least  $d$  conflict lists: for  $l = 1, 2, \dots, d$ , the point  $\pi(p)$  is dominated by some corner  $\pi(c_{j_l})$  of  $M_{i+l}$  such that  $\pi(c_{j_l})$  is dominated by  $\pi(c_j)$ . If  $\pi(c_{j_l})$  is dominated by  $\pi(c_j)$ , then  $z(c_{j_l}) > z(c_j)$ . Since  $p$  is in the conflict list of  $c_j$ ,  $z(p) < z(c_j)$  and  $z(p) < z(c_{j_l})$ . Hence  $p$  is contained in the conflict lists of at least  $d$  corners and  $p \in P_1$ .  $\square$

We can bound the number of points in conflict lists of corners with empty regions.

LEMMA 3.3. *The number of points in a set  $P_2$  is bounded by  $O(m/d)$ .*

*Proof.* We assign  $d$  dollars to every point in  $\text{list}(C_i)$  for every cell  $C_i$  of  $\mathcal{C}$ . The same point can appear in many lists, but the total number of elements in all conflict lists is  $O(m)$  and our total budget is  $O(m \cdot d)$  dollars. We assume that the cost of adding a point to  $P_2$  is  $d^2$

dollars, and we will show that  $m \cdot d$  dollars are sufficient to construct  $P_2$ . If the region of  $c_j$  is empty, then it contains at least  $d^2$  corners  $c_{ji}$ . We charge 1 dollar to every point in the conflict list of each  $C_{ji}$ . Every corner  $c_r$  is contained in at most  $d$  different regions: By definition of a region, a corner on  $M_i$  can be contained only in the region of a corner on  $M_j$  for  $i > j \geq i - d$ . Regions of corners on the same level of maxima are disjoint. Thus every point is charged at most  $d$  times. Hence  $P_2$  contains  $O(m/d)$  points.  $\square$

**Constructing the set  $P_3$ .** In the final step, we exclude rectangles that contain too many points from  $P$  from the set  $\mathcal{R}$ . Let  $list(\mathcal{C}) = \cup_{C_j \in \mathcal{C}} list(C_j)$  denote all points from  $P$  contained in the cells of  $\mathcal{C}$ . For any rectangle  $R \in \mathcal{R}$ ,  $|R \cap list(\mathcal{C})| \leq t \cdot d^2$ : a rectangle  $R$  is a subset of the non-empty region associated to some corner  $c_j$ . Hence every rectangle intersects projections of at most  $d^2$  different cells. Therefore  $R \cap list(\mathcal{C}) \leq t \cdot d^2$  for all  $R \in \mathcal{R}$ .

If  $|R \cap P| \geq t \cdot d^4$  for some rectangle  $R \in \mathcal{R}$ , we add all points of  $R \cap list(\mathcal{C})$  to  $P_3$  and remove the rectangle  $R$  from  $\mathcal{R}$ . We can show that  $|P_3| = O(m/d)$  using the same argument as in Lemma 3.3: We assign  $d$  dollars to each point of  $P$  and assume that the cost of inserting a point into  $P_3$  is  $d^2$  dollars. If points from  $R \cap list(\mathcal{C})$  are added to  $P_3$ , then we charge the cost to  $P \cap R$ . The cost is evenly distributed among all points of  $P \cap R$ . Since  $|P \cap R| \geq d^4 \cdot t$  and  $|list(\mathcal{C}) \cap R| \leq d^2 \cdot t$ , we charge no more than 1 dollar to each point.

All rectangles associated to corners of a fixed maximal layer  $M_i$  are disjoint. A planar point between the staircase of  $M_i$  and the staircase of  $M_{i+1}$  can be covered by at most one rectangle associated to some corner of  $M_j$  for every  $j$ ,  $i - d < j \leq i$ . Therefore  $\pi(p)$  for any point  $p \in P$  is contained in at most  $d$  rectangles from  $\mathcal{R}$ . Thus each point of  $P$  is charged at most  $d$  times. Hence the total cost of creating  $P_3$  is  $O(m \cdot d)$  dollars, where  $m$  is the total number of points in  $P$ . Since the cost of inserting a point into  $P_3$  is  $d^2$ ,  $P_3$  contains  $O(m/d)$  points.

Summing up,  $P' = P_1 \cup P_2 \cup P_3$  contains  $O(m/d)$  points. For every cell  $C_j$  of  $\mathcal{C}$ ,  $list(C_j) \cap (P \setminus P')$  is covered by  $O(d^3)$  rectangles from  $\mathcal{R}$  and every  $R \in \mathcal{R}$  contains  $O(t \cdot d^4)$  points of  $P$ .  $\square$

#### 4 Dominance Queries in a Slab

Now we describe the compact data structure that supports capped dominance range reporting queries. By a slight misuse of notation, in this section  $P$  will denote the set of points in a slab and  $m = |P|$ . We construct a  $t$ -shallow cutting with  $t = \log^6 n$  and apply Theorem 3.1 with  $d = \log n$ ; the subset  $P' \subset P$  and the set of rect-

angles  $\mathcal{R}$  are as defined in Theorem 3.1. We keep  $P'$  in a data structure from [5] that uses  $O(\log n)$  bits per point and answers queries in  $O(\log \log n + k)$  time. Let  $P_j = list(C_j) \setminus P'$  denote the set of points in the conflict list of the cell  $C_j$  that are not in  $P'$ . We construct a dominance data structure for points in  $P_j$ . Points in  $P_j$  are reduced to the rank space, so that the cell data structure uses  $O(\log t) = O(\log \log n)$  bits per point. As will be explained below, we can "decode" every point  $p$  in  $P_j$  (i.e., obtain the actual coordinates of  $p$ ) in  $O(\log^\epsilon n)$  time using additional  $O(\log t + \log d) = O(\log \log n)$  bits per point.

A  $(\log n)$ -capped dominance query is answered as follows: 1. we find the cell  $C_j$  that contains  $q$ . If  $C_j$  exists, it can be found in  $O(\log \log n)$  time by answering a point location query in a planar orthogonal subdivision of size  $O(m/t)$ . If  $q$  is not contained in any cell of the  $t$ -shallow cutting, then  $q$  dominates at least  $t = \log^6 n$  points and we can return *NULL*. 2. Then we reduce  $q$  to the rank space of  $P_j$ , and report all points in  $P_j$  that are dominated by  $q$ .  $P_j$  contains a poly-logarithmic number of points and each point of  $P_j$  can be accessed in time  $O(\log^\epsilon n)$ . Using standard techniques, we can reduce  $q$  to the rank space and answer a dominance query on  $P_j$  in  $O(t_{acc}(k+1))$  time. For completeness, we provide the details in Section 8. 3. We also query the data structure for  $P'$  and report all points in  $P'$  that are dominated by  $q$  in  $O(\log \log n + k)$  time.

It remains to show how the points in  $P_j$  can be "decoded", i.e., how to obtain the original coordinates of a point from its coordinates in the rank space of  $P_j$ . For each cell  $C_j$  of the shallow cutting we keep the list  $rlist(C_j)$  of rectangles  $R_{i_1}, R_{i_2}, \dots$ , such that  $list(C_j) \setminus P'$  is contained in these rectangles; for every rectangle we store the global coordinates of its endpoints. By Theorem 3.1  $rlist(C_j)$  consists of  $O(d^3)$  rectangles. We can identify a point  $p$  in  $list(C_j) \setminus P'$  by specifying the rectangle  $R_{i_p}$  that contains  $p$  and its  $x$ -rank in the rectangle  $R_{i_p}$  (the  $x$ -rank of a point  $p$  in a rectangle  $R$  is the number of points in  $R$  to the left of  $p$ ). Since the list  $rlist(C_j)$  consists of  $O(d^3)$  rectangles, the rectangle  $R_{i_p}$  can be identified by its position in  $rlist$  using  $O(\log d)$  bits. Each rectangle  $R \in \mathcal{R}$  contains  $O(t \cdot d^4)$  points. Hence we can store the  $x$ -rank of  $p$  in  $R_{i_p}$  using  $O(\log t + \log d)$  bits. Thus  $p$  is represented using  $O(\log t + \log d) = O(\log \log n)$  bits.

Every rectangle  $R \in \mathcal{R}$  is unbounded along the  $z$ -axis. Hence we can retrieve the point coordinates by answering a special kind of two-dimensional queries, further called *capped range selection* queries. A capped range selection query  $(Q, v)$  for a two-dimensional rectangle  $Q = [a, b] \times [c, d]$  and an integer  $v \leq \log^{10} n$  returns

the point with  $x$ -rank  $v$  in the rectangle  $Q$ , i.e., the  $v$ -th leftmost point in  $Q$ . Let  $P_{glob}$  denote the global set of points. Slabs are created by dividing the structure on previous recursion level along the  $x$ - or  $y$ -axis. Therefore  $P_{glob} \cap R = P \cap R$  for every  $R \in \mathcal{R}$ . Hence we need only one instance of the capped range selection data structure for all recursive slabs.

Capped range selection can be viewed as a generalization of the range successor problem [28] and can be solved using compact range trees and some auxiliary data structures. We will show in Section 6 how capped selection queries can be answered in  $O(\log^\varepsilon n)$  time using an  $O(n)$  space data structure. However, we have to decompose each rectangle from  $\mathcal{R}$  into  $O(\log^{1+\varepsilon} n)$  parts. Therefore we need  $O(\log^{2+\varepsilon} n)$  additional bits per rectangle. A detailed description is provided in Lemma 6.1 in Section 6.

Using Lemma 6.1, we can retrieve the coordinates of any point  $p$  in  $C_j$  in  $O(\log^\varepsilon n)$  time: we know the rectangle  $R \in \mathcal{R}$  that contains  $p$  and we know the  $x$ -rank  $\ell$  of  $p$  in  $R$ . Hence we can return the global coordinates of  $p$  in  $O(\log^\varepsilon n)$  time by answering a query  $(R, \ell)$ . If we can retrieve a point from cell  $C$  in time  $O(\log^\varepsilon n)$ , we can also reduce the query  $q$  to the rank space of  $C$  within the same time; see Section 8.

The conflict lists of all cells contain  $O(m)$  points. Data structures for all cells of the shallow cutting use  $O(m \log \log n)$  bits. For each cell we also store the list of rectangles and spend  $O(\log^{2+\varepsilon} n)$  bits per rectangle (for the capped selection data structure). In total we need  $O(d^3 \log^{2+\varepsilon} n) = o(t)$  bits for every cell and  $o(m)$  bits for all cells. The data structure for the subset  $P'$  uses  $O((m/\log n) \log n) = O(m)$  bits. To identify a cell of the shallow cutting that contains the query point, we need a data structure that supports point location queries on a planar orthogonal subdivision of size  $O(m/t)$  [5]. This data structure uses  $O(m/\log^5 n)$  bits. Hence our dominance data structure uses  $O(m \log \log n)$  bits in total. We need only one instance of the range selection data structure for all slabs; in this section we use the variant that consumes  $O(n \log n)$  bits of space. Hence the total space usage of the global data structure is not affected.

We also use range selection to access points stored in the slabs of size  $O(\log^8 n)$  of the grid structure. The number of points in every slab that is not divided further is  $\Omega(\log^4 n)$ . Since every point is stored in  $O(\log n / \log \log n)$  recursive structures, the number of slabs is bounded by  $O(n / \log^3 n)$ . Since the range selection data structure requires  $O(\log^{2+\varepsilon} n)$  bits per rectangle (i.e., per slab), the space usage of the range selection data structure is  $O(n)$ .

## Two-Dimensional Range Minima Queries.

Our data structure for five-sided queries can be easily modified to support range minima queries. We can adjust the slab dominance data structure, described in this section, so that it supports 2-d dominance minima queries. We use the same shallow cutting, but construct a data structure supporting 2-d dominance minimum queries for every cell. We replace the top data structure  $D^t$  with the data structure from [6]. This data structure contains  $O(n/\log n)$  points and uses  $O(n)$  space (on the top recursion level). We can decompose a four-sided minimum query into  $O(\log \log n)$  dominance minima queries and  $O(\log \log n)$  minima queries on top data structures as described in Section 2. Thus the answer to a (four-sided) range minimum query in 2-d is the minimum of the answers to  $O(\log \log n)$  local queries. The total query time is  $O(\log^\varepsilon n \log \log n)$ . We can get rid of the  $\log \log n$  term by replacing  $\varepsilon$  with an arbitrary  $\varepsilon' < \varepsilon$ .

## 5 Range Minima: Faster Queries in More Space

In this section we describe a data structure that uses  $O(n \log \log n)$  words of space and answers queries in  $O((k+1) \log \log n)$  time. We use the same recursive grid as in Section 2. Our approach is based on constructing a data structure for four-sided queries in every slab.

**LEMMA 5.1.** *There exists a data structure supporting capped four-sided queries in  $O(\log \log n + k)$  time and  $O(m \log m)$  bits of space where  $m$  is the number of points in a slab. The data structure relies on a universal data structure for two-dimensional range selection queries.*

*Proof.* W. l. o. g. we consider queries  $[a, b] \times [0, c] \times [0, d]$ . We construct a range tree with node degree  $\log^\varepsilon n$  on  $x$ -coordinates of points. We keep two dominance data structures in every tree node. These data structures support queries  $[0, b] \times [0, c] \times [0, d]$  and  $[a, +\infty) \times [0, c] \times [0, d]$ . Additionally each node contains a data structure that stores modified points: for every point  $p = (p_x, p_y, p_z)$  stored in a node  $u$ , we keep a point  $(i, p_y, p_z)$ , such that  $p$  is stored in the  $i$ -th child of  $u$ . This data structure supports "narrow" four-sided queries of the form  $[i_1 \times i_2] \times [0, c] \times [0, d]$  where  $1 \leq i_1 \leq i_2 \leq \log^\varepsilon n$ .

To answer a query we identify the leaves  $l_a$  and  $l_b$  holding the successor of  $a$  and the predecessor of  $b$  respectively. Let  $w$  denote the lowest common ancestor of these two leaves. Let  $w_i$  and  $w_j$  denote the children of  $w$  that are ancestors of  $l_a$  and  $l_b$ . We answer the query  $[a, +\infty) \times [0, c] \times [0, d]$  in  $w_i$  and  $[0, b] \times [0, c] \times [0, d]$  in  $w_j$ . Additionally we answer a narrow four-sided query  $[i+1, j-1] \times [0, b] \times [0, d]$  in the node  $w$ . The

answers to the three queries contain all points from  $[a, b] \times [0, c] \times [0, d]$ .

Dominance data structures are implemented as in Section 4 and use  $O(\log \log n)$  bits per point. We will show below that each narrow four-sided structure also uses  $O(\log \log n)$  bits per point. Since each point is stored twice on every level of the range tree and there are  $O(\log m / \log \log n)$  levels, the total space usage is  $O(m \log m)$  bits.  $\square$

Now we describe how “narrow” four-sided queries are answered. The following property of two-dimensional  $t$ -shallow cuttings, very similar to Theorem 3.1, will be used in our method. To keep the description unified with the rest of this section, we consider points on the  $(y, z)$ -plane.

**THEOREM 5.1.** *Let  $\mathcal{C}$  denote a  $t$ -shallow cutting of a two-dimensional set  $P$ ,  $|P| = m$ . There exists a subset  $P'$  of  $P$  and a set of 2-d rectangles  $\mathcal{R} = \{R_1, R_2, \dots, R_s\}$ , such that*

- (a)  $|P'| \leq m/d$
- (b) Rectangles  $R_i$  are unbounded along the  $z$ -axis.
- (c) The conflict list of any cell, except for points from  $P'$ , is contained in  $O(d)$  rectangles from  $\mathcal{R}$ ,

$$\begin{aligned} \text{list}(C_i) \cap (P \setminus P') \subseteq & (\text{list}(C_i) \cap R_{i_1}) \cup \\ & \cup (\text{list}(C_i) \cap R_{i_2}) \cup \dots \cup (\text{list}(C_i) \cap R_{i_g}) \end{aligned}$$

for  $g = O(d)$ .

- (d) Each rectangle contains  $O(t \cdot d)$  points of  $P$ .

*Proof.* For a cell  $C_i$  of the shallow cutting, let  $c_i$  denote its upper right corner. We can assume w.l.o.g. that no cell  $C_i$  is entirely contained in some other cell  $C_j$ . Hence no  $c_i$  is dominated by  $c_j$ . We will also assume that all corners  $c_i$  are sorted in increasing order by  $y$ -coordinates (and thus in decreasing order by  $z$ -coordinates).

We define  $P_1$  to be the set of points stored in more than  $d$  conflict lists. Since the total number of elements in all conflict lists is  $O(n)$ ,  $P_1$  contains  $O(n/d)$  points. The set  $\mathcal{R}$  contains a rectangle  $R_j = [c_{j-1}.y, c_j.y] \times (-\infty, +\infty)$  for every  $c_j$  (we set  $c_0.y = 0$ ). If a rectangle  $R_j$  contains over  $t \cdot d$  points, we add all points from  $R'_j = [c_{j-1}.y, c_j.y] \times [0, c_j.z]$  to  $P_2$  and remove  $R_j$  from  $\mathcal{R}$ . See Fig 3. All points in  $R'_j$  are dominated by  $c_j$ ; hence  $R'_j$  contains at most  $2t$  points. Therefore there are  $d/2$  points in  $P$  for every point in  $P_2$  and  $|P_2| \leq 2n/d$ . We set  $P' = P_1 \cup P_2$  so that  $|P'| = O(n/d)$ . Consider an arbitrary cell  $C_i$  and points dominated by  $c_i$ . Suppose that a point  $p \in P$ , dominated by  $c_i$ , is also dominated

by  $c_{i-l}$  for some  $l \geq d$ . Since  $z$ -coordinates of corners increase when  $i$  decreases,  $p$  is dominated by all  $c_j$ ,  $i-l \leq j \leq i$ . Hence  $p$  is contained in at least  $d$  cells and  $p \in P_1$ . Consider a point  $p \in R_j$  for  $i-d < j \leq i$ , such that  $p$  is dominated by  $c_i$ . If  $R_j$  contains over  $t \cdot d$  points, then  $p \in P_2$ .  $\square$

**LEMMA 5.2.** *There exists a data structure that answers narrow four-sided queries in  $O(\log \log n + k)$  time and uses  $O(m \log \log n)$  bits of space, where  $m$  is the number of points in the data structure.*

We combine the approach from [7] with a compact representation of two-dimensional shallow cuttings.

Let  $P_i$  denote the set of points  $p$  such that  $p.x = i$ . We consider projections of points in  $P_i$  onto the  $(y, z)$  plane and construct a 2-d  $t$ -shallow cutting for  $P_i$  with  $t = \log^3 n$ . Using Theorem 5.1, we construct a subset  $P'_i \subset P_i$  such that  $|P'_i| = O(|P_i|/d)$  for  $d = \log^2 n$ . For each cell  $C_j$ ,  $\text{list}(C_j) \setminus P'_i$  is contained in  $O(d)$  rectangles unbounded in  $z$ -direction.

All points from  $P'_i$  are stored in the data structure from [6] that uses  $O(|P'_i| \log^{1+\varepsilon} n)$  bits and supports queries in  $O(\log \log n + k)$  time. We follow the method of [7] to support four-sided narrow queries on  $\cup(P_i \setminus P'_i)$ . The only difference with [7] is that points in every group  $G$  (defined as in of [7]) are stored in the rank space. Additionally for every point  $p$  in  $G$  we store: (a) the index of  $i$  of  $P'_i$ , such that  $p \in P'_i$ , (b) the identifier of the rectangle  $R \in \mathcal{R}$  that contains  $p$ , (c) the  $y$ -rank of  $p$  in  $R$ , and (d) the cell  $C_f$ , such that  $p \in C_f$ .

Every group  $G$  contains points from  $O(\log^{2\varepsilon} n)$  different cells. Hence we can specify  $C_f$  using  $O(\log \log n)$  bits. Since  $\text{rlist}(C_f)$  consists of  $O(d^2)$  rectangles by Theorem 5.1, we can specify  $R$  using  $O(\log d) = O(\log \log n)$  bits. We can store the index of  $P_i$  in  $O(\log \log n)$  bits because a node has  $\log^\varepsilon n$  children. And we can store the  $y$ -rank of  $p$  in  $R$  using  $O(\log \log n)$  bits because the number of elements in  $R$  is polylogarithmic. Hence we spend  $O(\log \log n)$  bits per point. Additionally we store the list of rectangles  $\text{rlist}(C)$  for every cell  $C$ . All lists use  $O(m \cdot d \log^3 n / t) = O(m)$  bits.

We also need an  $O(n \log \log n)$ -word universal data structure for capped selection, implemented as in Lemma 6.1, part (b). When this data structure and the above information are available, we can retrieve the coordinates of a point in  $O(\log \log n)$  time. Thus, as explained in Section 8, we can support queries on the rank-reduced points of  $G$  in  $O(\log \log n)$  time per reported point. A general narrow four-sided query can be reduced to a query on a group  $G$  [7]. Hence, we can answer narrow four-sided queries in  $O((k+1) \log \log n)$  time.

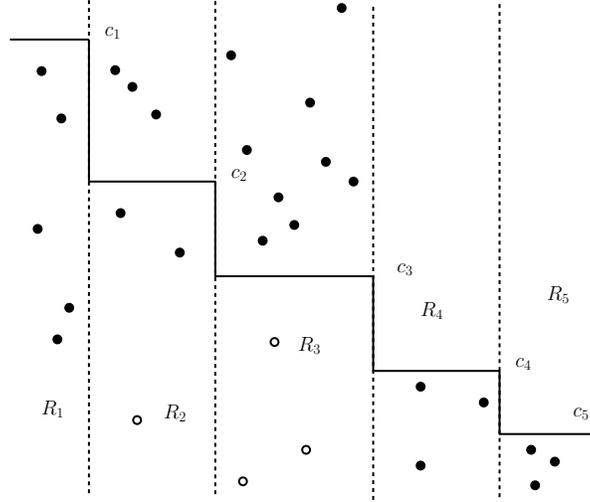


Figure 3: Example of a  $t$ -shallow cutting in two dimensions and its covering for  $t = 3$  and  $d = 3$ . Points from  $P'$  are depicted by hollow circles, all other points are depicted by filled circles. Dashed lines are boundaries of rectangles  $R_i$ . The hollow point in rectangle  $R_2$  must be stored in conflict lists of  $C_2, C_3$ , and  $C_4$ . Therefore this point is in  $P_1$ . Hollow points in rectangle  $R_3$  are in  $P_2$  because  $R_3$  contains over  $t \cdot d$  points.

**THEOREM 5.2.** *There exists a data structure that supports five-sided three-dimensional range reporting queries in  $O((k + 1) \log \log n)$  time and uses  $O(n \log \log n)$  space.*

*The same data structure can be adjusted to support two-dimensional range minima queries in  $O(\log \log n)$  time.*

*Proof.* We use the recursive grid described in Section 2 and store the four-sided data structure for each slab. Points in every slab are reduced to the rank space. Each point in a slab can be "decoded" in  $O(\log \log n)$  time. Hence we can transform a query to the rank space of a slab in  $O(\log \log n)$  time.

A capped five-sided query is processed as follows. If a query is entirely contained in one horizontal or vertical slab, we answer the query using the data structure for that slab. Otherwise a capped five-sided query can be represented as a union of at most four capped four-sided queries and a query to the top data structure. Hence a capped query is answered  $O((k + 1) \log \log n)$  time.

If  $k > \log n$ , we use the data structure of Chazelle [9] that uses  $O(n \log \log n)$  words and supports queries in  $O(\log n \log \log n + k \log \log n)$  time. If  $k \geq \log n$ , the query time of Chazelle's structure can be simplified to  $O(k \log \log n)$ . As explained in Section 4, we can adjust our result to support range minima queries in time  $O(\log \log n)$ .

The space usage of the data structure in bits is  $S(n) = O(n \log n) + 2\sqrt{n/\log^2 n} nS(\sqrt{n \log^2 n})$ . Hence  $S(n) = O(n \log n \log \log n)$ , see e.g., [2, 24]. Since the data structure of Chazelle [9] also needs  $O(n \log \log n)$

words, the total space usage is  $O(n \log \log n)$  words.  $\square$

The result of Theorem 5.2 can be extended to general (six-sided) three-dimensional range reporting. We use the standard approach of constructing a range tree with node degree  $\log^\varepsilon n$  and storing the data structure of Theorem 5.2 in each node. The space usage increases by the factor of  $O(\log n / \log \log n)$  and the query time is not changed.

**COROLLARY 5.1.** *There exists a data structure that supports orthogonal three-dimensional range reporting queries in  $O((k + 1) \log \log n)$  time and uses  $O(n \log n)$  space.*

*The same data structure can be adjusted to support three-dimensional range emptiness queries in  $O(\log \log n)$  time.*

## 6 Data Structure for Capped Selection Queries

**LEMMA 6.1.** *Let  $P$  be a set of  $n$  two-dimensional points and  $\mathcal{R}$  be a set of  $m_R$  rectangles. There exists a data structure that uses  $O(s(n) + m_R \log^{1+\varepsilon} n)$  words and answers capped range selection queries  $(R, f)$  where  $R \in \mathcal{R}$  and  $f = O(\log^{10} n)$ , in time  $q(n)$ . The following trade-offs between  $q(n)$  and  $s(n)$  are possible:*

- (a)  $s(n) = O(n)$  and  $q(n) = O(\log^\varepsilon n)$
- (b)  $s(n) = O(n \log \log n)$  and  $q(n) = O(\log \log n)$
- (c)  $s(n) = O(n \log^\varepsilon n)$  and  $q(n) = O(1)$

We store a compact range tree with node degree

$\log^\varepsilon n$  for some constant  $\varepsilon > 0$  on a set  $P$ . Recall that a standard range tree is a balanced tree on the  $x$ -coordinates of points. Every internal node stores the set of points  $S(u)$  that contains all points whose  $x$ -coordinates are in the leaf descendants of  $u$ . Although a compact range tree does not store  $S(u)$  in explicit form, it supports operations `noderange`( $y_1, y_2, u$ ) and `point`( $u, i$ ). The former operation identifies the maximum range  $[h_1..h_2]$  such that all points stored in  $S(u)$  at positions  $h_1, h_1 + 1, \dots, h_2$  have  $y$ -coordinates in the interval  $[y_1, y_2]$ . The operation `point`( $u, i$ ) returns the coordinates of the  $i$ -th point in  $S(u)$  (assuming that points in  $S(u)$  are sorted by their  $y$ -coordinates). Different trade-offs between the space usage  $s_{\text{comp}}(n)$  of the compact tree and the time  $t_{\text{comp}}(n)$  of `point`( $u, i$ ) are possible: either (i)  $s_{\text{comp}}(n) = O(n)$  and  $t_{\text{comp}}(n) = O(\log^\varepsilon n)$  or (ii)  $s_{\text{comp}}(n) = O(n \log \log n)$  and  $t_{\text{comp}}(n) = O(\log \log n)$  or (iii)  $s_{\text{comp}}(n) = O(n \log^\varepsilon n)$  and  $t_{\text{comp}}(n) = O(1)$ .

For every node in the range tree we store a data structure supporting range  $\tau$ -selection queries: for any  $y$ -range  $[y_1, y_2]$  and any  $f \leq \tau$ , we can return the index of the point with the  $f$ -th smallest  $x$ -coordinate in  $S(u)[y_1..y_2]$ . When  $\tau = \log^{O(1)} n$ , we can support range  $\tau$ -selection queries in  $O(1)$  time using  $O(\log \log n)$  bits per point [21, 17].

Every covering rectangle  $R = [a, b] \times [c, d]$  is divided into  $O(\log^{1+\varepsilon} n)$  smaller rectangles: we can represent  $[a, b]$  as a union of  $O(\log^{1+\varepsilon} n)$  intervals  $[a_i, b_i]$  where  $a_i$  and  $b_i$  are the leftmost and the rightmost leaf descendants of some node  $u_i$  in the range tree. Let  $[c_i, d_i] = \text{noderange}(c, d, u_i)$ . We store the coordinates of each  $R_i = [a_i, b_i] \times [c_i, d_i]$ , and the number of points  $f_i = |P \cap R_i|$  for each  $R_i$ . We also compute the prefix sums  $F_i = \sum_{j=1}^i f_j$  for all  $i$ .

A query  $(R, f)$  is answered as follows. We consider the decomposition of  $R$  into rectangles  $R_i$  and find the index  $j$ , such that  $F_{j-1} < f \leq F_j$ . Let  $f' = f - F_{j-1}$ . Using the range selection data structure, we can find the index of the  $f'$ -th leftmost point in  $S(u)[c'..d']$  where  $[c', d'] = \text{noderange}(c, d, u)$ . Then we can obtain the point by answering the `point` query.

The query time is dominated by the operations on the range tree. Hence we obtain the same space-time trade-offs for the capped range selection as for the compact range tree.

## 7 Four-Dimensional Dominance Range Reporting

In this section we describe a data structure that answers dominance range reporting queries in  $O(\log n / \log \log n + k)$  time. We start by explaining why previous methods require  $\Omega(\log n)$  time. Then

we describe the main idea of our approach and show how it can be used to reduce the query time by a factor  $O(\sqrt{\log \log n})$ . Then we will present a complete solution.

**Range Trees.** Four-dimensional queries are reduced to three-dimensional queries using a range tree. A range tree  $T$  stores the points of the input set  $P$  sorted by their fourth coordinate. In every internal node  $u$  of the range tree, we keep a set of points  $S(u)$ ;  $S(u)$  contains all points stored in the leaf descendants of  $u$ . In every node  $u$  we keep a data structure supporting three-dimensional dominance queries. Let  $\gamma$  denote the node degree of the range tree. For any interval  $[a, b]$ , we can identify  $O(\gamma \cdot \log_\gamma n)$  nodes  $u_i$  of the range tree, such that  $p_{z'} \in [a, b]$  if and only if  $p \in S(u_i)$ . Furthermore nodes  $u_i$  can be divided into  $O(\log_\gamma n)$  groups, such that nodes in the same group are siblings. Nodes  $u_i$  will be called canonical nodes for an interval  $[a, b]$  and sets  $S(u_i)$  will be called a canonical decomposition of  $[a, b]$ . All points dominated by a query point  $q = (q_x, q_y, q_z, q_{z'})$  are reported as follows: We visit all canonical nodes  $u_i$  for  $[1, q_{z'}]$  and report, for each  $u_i$ , all points  $p \in S(u_i)$  dominated by  $(q_x, q_y, q_z)$ . That is, we answer three-dimensional dominance queries in the canonical nodes for  $[1, q_{z'}]$ .

A three-dimensional dominance query can be answered by locating a point in a planar orthogonal subdivision [22, 1, 5]. Orthogonal point location queries can be answered in  $O(\log \log n)$  time [5]. If the range tree  $T$  is a binary tree, we can answer four-dimensional dominance queries in  $O(\log n \log \log n + k)$  time. By increasing the node degree  $\gamma$  to  $O(\log^\varepsilon n)$ , we can reduce the query time to  $O(\log n + k)$ : for every range of sibling nodes  $u_1, \dots, u_r$  in the range tree, we keep a three-dimensional data structure on points from  $S(u_1) \cup \dots \cup S(u_r)$ . This reduces the number of sets in the canonical decomposition of  $[1, q_{z'}]$  to  $O(\log_\gamma n) = O(\log n / \log \log n)$ . But each point must be stored in  $O(\log_\gamma n \cdot \gamma^2)$  three-dimensional data structures. Unfortunately it appears that further improvement in query time is not possible with this approach:  $O(\log \log n)$  query time is optimal for both planar orthogonal point location and three-dimensional reporting queries. This lower bound is valid for any data structure with pseudo-linear space usage and follows from the lower bound for the predecessor problem [4, 31]. Increasing the node degree is also not feasible: every point must be stored in  $\Omega(\gamma^2)$  three-dimensional data structures. Hence if  $\gamma = \log^{\omega(1)} n$ , the total space usage would be prohibitively high.

**Better Query Time.** In order to improve the query time we store additional information for selected nodes in the range tree. Our range tree  $T$  has node

degree  $\gamma = \log^\varepsilon n$ . We embed a tree  $T^0$  with node degree  $\rho_0 = \gamma^{\alpha_0}$  where  $\alpha_0 = \log \log n$  into  $T$ . Nodes of  $T^0$ , further called 0-nodes, correspond to nodes of  $T$  with depth that is divisible by  $\log \log n$ . For every range  $1 \leq f \leq l \leq \rho_0$  we define  $S(u, f, l) = \cup_{j=f}^l S(u_j)$  where  $u_j$  denotes the  $j$ -th child of a node  $u$ . Sets  $S(u, \cdot, \cdot)$  will be called the node ranges of  $u$ . Let  $t_0 = \rho_0^4$ . We construct a  $2t_0$ -shallow cutting  $\mathcal{C}(u, i, j)$  for each  $S(u, i, j)$  and for every internal node with height at least 2 in the tree  $T^0$ . We also construct a  $2t_0$ -shallow cutting  $\mathcal{E}(v, i, j)$  for each set  $S(v, i, j) = \cup_{l=i}^j S(v_l)$  where  $v$  is a node in  $T$  and  $v_l$  is the  $l$ -th child of  $v$ . Finally, we construct a  $t_h$ -shallow cutting  $\mathcal{C}'(E_j)$  for each cell  $E_j$  of  $\mathcal{E}(v, i, j)$ , where  $t_h = \gamma^{O(1)}$ .

The set  $S(u, i, j)$  of a 0-node  $u$  can be represented as a union of  $O(\log \log n)$  non-overlapping sets  $S(v_f, i_f, j_f)$  where each node  $v_f$  is "between" the node  $u$  and its children in  $T^0$ . In other words, each  $v_f$  is a descendant of  $u$  and an ancestor of at least one  $u_l$ ,  $i \leq l \leq j$ . We will say that sets  $S(v_f, i_f, j_f)$  are a *canonical decomposition* of  $S(u, i, j)$ .

**LEMMA 7.1.** ([25]) *Let  $\mathcal{A}$  be an  $f$ -shallow cutting for a set  $S$  and let  $\mathcal{B}$  be an  $(f')$ -shallow cutting for a set  $S' \subseteq S$  so that  $f' \geq 2f$ . Every cell  $A_i$  of  $\mathcal{A}$  is contained in some cell  $B_j$  of  $\mathcal{B}$ .*

Consider a set  $S(u, i, j)$  and its canonical decomposition  $S(u, i, j) = \cup_f S(v_f, i_f, j_f)$ . By Lemma 7.1, a cell of  $\mathcal{C}_j$  of  $\mathcal{C}(u, i, j)$  is contained in some cell  $E_{j_f}$  of  $\mathcal{E}(v_f, i_f, j_f)$ . For each cell  $C_l$  of a shallow cutting  $\mathcal{C}(u, i, j)$  and for every set in the canonical decomposition of  $S(u, i, j)$  we store a pointer to a cell  $E_{l_f} \in \mathcal{E}(v_f, i_f, j_f)$  containing  $C_l$ .

**LEMMA 7.2.** ([5]) *There exists a data structure that answers point location queries in a two-dimensional orthogonal subdivision of a  $U \times U$  grid by  $O(m)$  rectangles in time  $O(\min(\sqrt{\log_\gamma m}, \log \log_\gamma U))$ .*

*Proof.* Using the predecessor data structure, we can reduce the point location problem to the special case when point coordinates are bounded by  $O(m)$ . Using the result of Chan [5] we can answer point location queries on an  $O(m) \times O(m)$  grid in  $O(\log \log_\gamma m)$  time<sup>4</sup>. Predecessor queries can be answered in time  $O(\sqrt{\log_\gamma m})$ . Hence the total query time is  $O(\sqrt{\log_\gamma m} + \log \log_\gamma m) = O(\sqrt{\log_\gamma m})$ .  $\square$

Now a query can be answered as follows. Suppose that we must report all points dominated by  $q =$

<sup>4</sup>The data structure described in [5] supports queries in  $O(\log \log m)$  time. A straightforward extension of this data structure supports queries in  $O(\log \log_\gamma m)$  time; see e.g., [7].

$(q_x, q_y, q_z, q_{z'})$ . We visit the nodes  $u$  of  $T^0$  on the path from the root to  $q_{z'}$ . In every visited node  $u$  we identify the canonical set  $S(u, i, j)$  and find the cell  $C_l$  of the shallow cutting  $\mathcal{C}(u, i, j)$  that contains  $q' = (q_x, q_y, q_z)$ . See Fig. 4 for an example. Consider the canonical decomposition  $S(u, i, j) = \cup_f S(v_f, i_f, j_f)$ . For every  $f$ , we visit the cell  $E_{l_f}$  of  $\mathcal{E}(v_f, i_f, j_f)$  that contains  $C_l$ . We identify the cell  $C'_f$  of the shallow cutting  $\mathcal{C}'(E_{l_f})$  that contains  $q$ . Every cell of  $\mathcal{E}(v_f, i_f, j_f)$  contains  $\gamma^{O(\log \log n)}$  points. Hence, we can answer a point location query and find the cell  $C'_f$  in  $O(\sqrt{\log \log n})$  time by Lemma 7.2. Since  $C'_f$  contains  $O(\log^2 n)$  points, we can answer a three-dimensional query on  $C'_f$  in  $O(1 + k)$  time; see Section 8, Lemma 8.2. If  $q'$  is not contained in any cell of  $C'$ , then the query range contains  $k \geq \log^2 n$  points and we can answer the query using some of the previously known data structures in time  $O(\log n + k) = O(k)$ .

Our procedure visits  $O(\frac{\log n}{(\log \log n)^2})$  nodes of  $T^0$  and  $O(\frac{\log n}{\log \log n})$  nodes of  $T$ . We spend  $O(\log \log n)$  time in every visited node of  $T^0$  and  $O(\sqrt{\log \log n})$  time in every visited node of  $T$ . Hence the total query time is  $O(\log n / \sqrt{\log \log n})$ .

**Optimal Query Time.** In order to further improve the query time, we embed a sequence of subtrees  $T^i$ , for  $i = 0, \dots, h$ , into  $T$ . Node degrees of these subtrees decrease exponentially. As above, we keep shallow cuttings for node ranges in every tree. A shallow cutting in a node range of  $T^i$  provides us with a clue (via Lemma 7.1) that speeds up the queries in  $T^{i+1}$ .

Let  $\alpha_0 = \log \log n$  and  $\alpha_i = (\alpha_{i-1})^{1/2} \log^2 \alpha_{i-1}$ . If  $u$  is a node in  $T^i$  and  $v$  is its child in  $T^i$ , then  $\alpha_i$  is the distance between  $u$  and  $v$  in  $T$ . Thus every node of  $T^i$  corresponds to subtree of height  $\alpha_i$  in  $T$ . We set  $\rho_i = \gamma^{\alpha_i}$  and  $t_i = (\rho_i)^4$ . We choose  $h$  so that  $\alpha_h = \Theta(1)$ . To avoid clumsy notation, we assume that  $\alpha_i$  divides  $\alpha_{i-1}$  for  $i \geq 1$  and  $\alpha_0$  divides the height of  $T$ . Nodes of  $T^i$  will be called  $i$ -nodes.

For every node  $u$  of  $T^i$ ,  $0 \leq i \leq h$ , and for every range of children  $S(u, l, r) = \cup_{j=l}^r S(u_j)$ , we construct a  $t_i$ -shallow cutting  $\mathcal{C}'(u, l, r)$ . For every node  $u$  of  $T^i$ ,  $1 \leq i \leq h$ , and for every range of children  $S(u, l, r)$ , we construct a  $2t_{i-1}$ -shallow cutting  $\mathcal{E}(u, l, r)$ . For each cell  $E$  of  $\mathcal{E}(u, l, r)$  we construct a  $t_i$ -shallow cutting  $\mathcal{C}'(E)$ . Consider a canonical decomposition of  $S(u, l, r)$  into  $S(v_f, l_f, r_f)$ ,  $S(u, l, r) = \cup S(v_f, l_f, r_f)$ , where  $u$  is a node of  $T^0$  and  $v_f$  are nodes of  $T^1$ . For each cell  $C'$  of  $\mathcal{C}'(u, l, r)$  and for every set  $S(v_f, l_f, r_f)$  in the canonical decomposition, we keep the pointer to a cell  $E_f$  of  $\mathcal{E}(v_f, l_f, r_f)$  such that  $E_f$  contains  $C'$ . Consider a canonical decomposition of  $S(u, l, r)$  into  $S(v_f, l_f, r_f)$ ,  $S(u, l, r) = \cup_f S(v_f, l_f, r_f)$ , where  $u$  is a node of  $T^i$  for

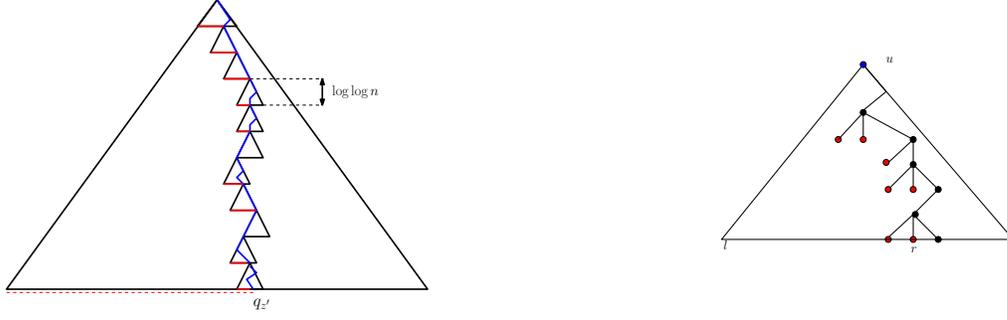


Figure 4: Left: Range tree and a path to  $q'_z$ . Triangles are subtrees corresponding to nodes of  $T^0$ . Red areas show the canonical decomposition of the  $[1, q'_z]$ . Right: A subtree corresponding to a node  $u \in T^0$  and its children. Red nodes are a canonical decomposition of the node range  $S(u, l, r)$ .

some  $i$ ,  $1 \leq i < h$  and  $v_f$  are nodes of  $T^{i+1}$ . For each cell  $C'$  of  $\mathcal{C}'(E)$ , where  $E$  is a cell of  $\mathcal{E}(u, l, r)$ , and for every set  $S(v_f, l_f, r_f)$  in the canonical decomposition, we keep the pointer to a cell  $E_f$  of  $\mathcal{E}(v_f, l_f, r_f)$  such that  $E_f$  contains  $C'$ .

Consider a canonical decomposition of  $S(u, l, r)$  into  $S(v_f, l_f, r_f)$ ,  $S(u, l, r) = \cup_f S(v_f, l_f, r_f)$ , where  $u$  is a node of  $T^h$ . For each cell  $C'$  of  $\mathcal{C}'(E)$ , where  $E$  is a cell of  $\mathcal{E}(u, l, r)$ , and for every set  $S(v_f, l_f, r_f)$  in the canonical decomposition of  $S(u, l, r)$ , we keep the pointer to a cell  $C_f$  of  $\mathcal{C}(v_f, l_f, r_f)$  such that  $C_f$  contains  $C'$ . We remark that the nodes in the canonical decomposition of  $S(u, l, r)$  are the nodes of  $T$ . We can adjust the constant  $\gamma = \log^\varepsilon n$  in such way, that  $t_h = \log^3 n/2$ . Hence, by Lemma 7.1, each cell  $C'$  is contained in some  $C_f$ .

A query is answered as follows. The set  $\pi^i$  consists of all  $i$ -nodes  $u$  on the path from the root to  $q'_z$ , such that the height of  $u$  is at least  $\ell = 3 \log \log n$ . A query is processed in  $h + 1$  stages. During stage  $i$  we visit nodes on  $\pi^i$ ; for every visited node we find the cell of  $\mathcal{C}'(E)$  that contains  $q' = (q_x, q_y, q_z)$ .

**Stage 0.** We visit nodes of  $T^0$  on the path  $\pi^0$ . In every visited node  $u$  we find the canonical set  $S(u, l, r)$  in the canonical decomposition of  $[0, q'_z]$ . Next we find the cell  $C'$  of  $\mathcal{C}'(u, l, r)$  that contains  $q$ . For every set  $S(v_f, l_f, r_f)$  in the canonical decomposition of  $S(u, l, r)$ , we visit the cell  $E_f$  of  $\mathcal{E}(v_f, l_f, r_f)$  that contains  $C$ . Then we locate the cell  $C'_f$  of  $\mathcal{C}'(E_f)$  that contains  $q$ .

**Stage  $i$ ,**  $1 \leq i \leq h$ . Consider the canonical decompositions of  $[0, q'_z]$  into node ranges  $S(u, l, r)$  for nodes  $u \in T^i$  and for some fixed  $i$ ,  $1 \leq i \leq h$ . Suppose that we already know the cell  $C'$  of  $\mathcal{C}'(E)$  that contains  $q'$  in every  $i$ -node  $u$  on  $\pi^i$ , where  $E$  is the cell of  $\mathcal{E}(u, l, r)$ . For every node range  $S(u, l, r)$  we consider its decomposition into  $(i + 1)$ -nodes  $S(u, l, r) = \cup S(v_f, l_f, r_f)$ . For each  $\mathcal{E}(v_f, l_f, r_f)$  we visit the cell  $E_f$  that contains  $C'$ ; then we locate the cell  $C'_f$  of  $\mathcal{C}'(E_f)$  that contains  $q$ .

**Final Step.** Consider the canonical decompositions of

$[0, q'_z]$  into node ranges  $S(u, l, r)$  for nodes  $u \in T^h$ . Suppose that we already know the cell  $C'$  of  $\mathcal{C}'(E)$  that contains  $q'$  in every  $h$ -node  $u$  on  $\pi^h$ . For every node range  $S(u, l, r)$  we consider its decomposition into  $(i + 1)$ -nodes  $S(u, l, r) = \cup S(v_f, l_f, r_f)$ . For each  $\mathcal{C}(v_f, l_f, r_f)$  we visit the cell  $C_f \in \mathcal{C}(v_f, l_f, r_f)$  that contains  $C'$ . Finally we report all points in  $list(C_f)$  that are dominated by  $q'$ .

Now we analyze the query time. To simplify the notation let  $\lambda(n) = \log n / \log \log n$  and let  $t_0 = n$ . Our method visits  $O(\lambda(n)/\alpha_i)$   $i$ -nodes. The time spent in a visited node is dominated by the time needed to answer a point location query on a set of  $O(t_{i-1})$  rectangles. Using the result of Chan [5] we spend  $O(\log \log n)$  time in each 0-node and  $O(\lambda(n))$  time in all 0-nodes. By Lemma 7.2, we spend  $O(\sqrt{\alpha_{i-1}})$  time in every  $i$ -node where  $1 \leq i \leq h$ . Hence the total time in all  $i$ -nodes is  $O(\lambda(n)\sqrt{\alpha_{i-1}}/\alpha_i) = O(\lambda(n)/\log^2(\alpha_{i-1}))$ .

LEMMA 7.3.  $\sum_{i=1}^h \frac{1}{\log^2 \alpha_{i-1}} = O(1)$

*Proof.* We will prove that  $S = \sum_{i=0}^h \frac{1}{\log^2 \alpha_i} = O(1)$ . We define the sequence  $f(i)$  as follows:  $f(0) = 0$ ,  $f(i) = \min\{x \mid \alpha_x \leq \log(\alpha_{f(i-1)})\}$ . Let  $\sigma_i = \sum_{j=f(i)}^{f(i+1)-1} \frac{1}{\log \alpha_j}$ . The sum  $\sigma_i$  has  $O(\log \log \alpha_{f(i)})$  terms. By definition of  $f()$ ,  $\alpha_j > \log(\alpha_{f(i)})$  for  $f(i) \leq j < f(i + 1)$ . Hence each term in  $\sigma_i$  is smaller than  $\frac{1}{(\log \log \alpha_{f(i)})^2}$  and  $\sigma_i = O(\frac{1}{\log \log \alpha_{f(i)}})$ .

We can represent  $S$  as the sum of  $\sigma_j$ . By the above analysis  $S = O(\sum_{f(i) \leq h} \frac{1}{\log \log \alpha_{f(i)}})$ . Let  $l$  denote the number of terms in the latter sum. Let  $\beta_i = \alpha_{f(i)}$ . Then  $S = O(S')$  where  $S' = \sum_{i=0}^{l-1} \frac{1}{\log \log \beta_{i-1}}$ . By definition of  $f(i)$ ,  $\beta_i \leq \log \beta_{i-1}$  and  $\log \log \beta_i < (1/2) \log \log \beta_{i-1}$ . Hence  $\frac{1}{\log \log \beta_i} > \frac{2}{\log \log \beta_{i-1}}$ . Since  $\beta_l = O(1)$ ,  $\frac{1}{\log \log \beta_l} = O(1)$ . Hence the sum  $S'$  can be bounded by a decreasing geometric sequence with constant first term. Therefore  $S = O(S') = O(1)$ .  $\square$

By Lemma 7.3, the total time that we need to locate  $q'$  in the shallow cuttings of all relevant nodes is  $O(\lambda(n))$ . A three-dimensional query on a cell  $C_f$  of a  $t$ -shallow cutting takes time  $O(1)$  (ignoring the time to report points).

One technicality still needs to be addressed. We must consider the nodes  $u$  on the path from the root to  $q'_z$ , such that the height of  $u$  is less than  $\ell$ . We answer a three-dimensional query in every such node in  $O(\log \log n)$  time. Since the number of such nodes is  $\ell = O(\log \log n)$ , the total query cost increases by a negligible term  $O((\log \log n)^2)$ .

Now we analyze the space usage. All auxiliary shallow cuttings  $\mathcal{E}(\cdot)$  and  $\mathcal{C}'(\cdot)$  use linear space: consider a node  $u$ . For every node range  $S(u, l, r)$  we store two shallow cuttings that have  $O(|S(u, l, r)|/t_i)$  and  $O(|S(u, l, r)|/t_{i-1})$  cells respectively. We store  $\alpha_i/\alpha_{i+1} < \alpha_i$  pointers for each cell of  $\mathcal{C}'(E)$ . Since each point of  $S(u)$  occurs in  $\rho_i^2$  node ranges the total space used by all shallow cuttings associated to the  $i$ -node  $u$  is  $O(m \cdot (\alpha_i/\rho_i^2)) = O(m/\rho_i)$  where  $m$  is the number of points in  $S(u)$ . Thus shallow cuttings in all nodes of  $T^i$  consume  $O((n \log n)/\rho_i)$  space. Since  $\sum_{i=0}^h (1/\rho_i) \log n = O(1)$ , all additional shallow cuttings consume  $O(n)$  space. Finally we can store points in the cells of shallow cuttings  $C_i(u, l, r)$  for all nodes  $u \in T$  in  $O(n \log^\varepsilon n)$  words of space using the method of [25]. Hence the total space usage is  $O(n \log^\varepsilon n)$ .

**THEOREM 7.1.** *There exists an  $O(n \log^\varepsilon n)$ -word data structure that answers four-dimensional dominance range reporting queries in  $O(\log n / \log \log n + k)$  time.*

We can use the same data structure to support five-sided four-dimensional queries, i.e., four-dimensional queries that are bounded on five sides. Using standard techniques this result can be extended to a data structure that uses  $O(n \log^{3+\varepsilon} n)$  space and answers arbitrary four-dimensional orthogonal range reporting queries in  $O(\log n / \log \log n + k)$  time.

**THEOREM 7.2.** *There exists an  $O(n \log^{3+\varepsilon} n)$ -word data structure that answers four-dimensional range reporting queries in  $O(\log n / \log \log n + k)$  time.*

It is possible to reduce the space usage of the data structure in Theorem 7.2 to  $O(n \log^{2+\varepsilon} n)$  words using the lopsided grid approach from [6]. This result is omitted from this paper for space reasons and will be presented in the full version of this paper [27].

## 8 Reduction to Rank Space and Range Reporting on a Small Set of Points

We can reduce an orthogonal range searching problem on a set  $P$  of  $m$  points to the special case when all point

coordinates are positive integers bounded by  $m$  [16, 2]. This can be achieved by replacing every point coordinate by its rank. In the case of three-dimensional points every point  $p = (p.x, p.y, p.z)$  in a set  $P$  is replaced with  $p' = (\text{rank}(p.x, S_x), \text{rank}(p.y, S_y), \text{rank}(p.z, S_z))$ , where  $P_x, P_y,$  and  $P_z$  denote the sets of  $x$ -,  $y$ -, and  $z$ -coordinates of points in  $P$ . For any point  $p$  we have:

$$p \in [a, b] \times [c, d] \times [e, f] \Leftrightarrow p' \in [a', b'] \times [c', d'] \times [e', f']$$

where  $a' = \text{rank}(\text{succ}(a, P_x), P_x)$ ,  $c' = \text{rank}(\text{succ}(c, P_y), P_y)$ ,  $e' = \text{rank}(\text{succ}(e, P_z), P_z)$ ,  $b' = \text{rank}(b, P_x)$ ,  $d' = \text{rank}(d, P_y)$ ,  $f' = \text{rank}(f, P_z)$ ; the successor of a value  $x$  in a set  $X$ , denoted  $\text{succ}(x, X)$ , is the smallest element in a set  $X$  that is larger than or equal to  $x$ .

The following Lemma can be used for rank reduction on a set of poly-logarithmic size.

**LEMMA 8.1.** [18] *Suppose that we can access any element of an integer set  $S$  in time  $O(t_{\text{acc}})$  and  $|S| = \log^{O(1)} n$ . There is a data structure that answers predecessor and successor queries on  $S$  in  $O(1 + t_{\text{acc}})$  time and uses  $O(|S| \log \log n)$  additional bits.*

Suppose that we store the set  $P$  of three-dimensional points such that  $|P| = \log^{O(1)} n$  and every point of  $P$  can be accessed in time  $O(t_{\text{acc}})$ . By Lemma 8.1, we can answer successor queries on  $P_x, P_y,$  and  $P_z$  in  $O(t_{\text{acc}})$  time using  $O(\log \log n)$  bits per point.

**LEMMA 8.2.** *If a set  $P$  contains  $m = \log^{O(1)} n$  points in the rank space of  $P$ , then we can keep  $P$  in a data structure that uses  $O(m \log \log n)$  bits and answers three-dimensional dominance range reporting queries and three-dimensional five-sided range reporting queries in  $O(k)$  time. This data structure uses a universal look-up table of size  $o(n)$ .*

Lemma 8.2 can be proved in exactly the same way as Lemma 7 in [26]. Lemma 7 in [26] is proved for dominance queries on  $O(\log^2 n)$  points. However exactly the same method can be also used for five-sided queries and for any poly-logarithmic number of points.

Combining Lemma 8.2 and the observations after Lemma 8.1, we have the following result.

**LEMMA 8.3.** *If a set  $P$  contains  $t = \log^{O(1)} n$  points and we can obtain the coordinates of any point in  $P$  in time  $O(t_{\text{acc}})$ . There is a data structure that uses  $O(m \log \log n)$  bits additional bits and answers three-dimensional dominance range reporting queries and three-dimensional five-sided range reporting queries in  $O(k \cdot t_{\text{acc}})$  time. This data structure uses a universal look-up table of size  $o(n)$ .*

## References

- [1] Peyman Afshani. On dominance reporting in 3d. In *Proc. 16th Annual European Symposium on Algorithms (ESA)*, pages 41–51, 2008.
- [2] Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. New data structures for orthogonal range searching. In *Proc. 41st Annual Symposium on Foundations of Computer Science, (FOCS)*, pages 198–207, 2000.
- [3] Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. Optimal static range reporting in one dimension. In *Proc. 33rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 476–482, 2001.
- [4] Paul Beame and Faith E. Fich. Optimal bounds for the predecessor problem and related problems. *Journal of Computer and System Sciences*, 65(1):38–72, 2002.
- [5] 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.
- [6] Timothy M. Chan, Kasper Green Larsen, and Mihai Patrascu. Orthogonal range searching on the RAM, revisited. In *Proc. 27th ACM Symposium on Computational Geometry, (SoCG)*, pages 1–10, 2011.
- [7] Timothy M. Chan, Yakov Nekrich, Saladi Rahul, and Konstantinos Tsakalidis. Orthogonal point location and rectangle stabbing queries in 3-d. In *Proc. 45th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 31:1–31:14, 2018.
- [8] Bernard Chazelle. Filtering search: a new approach to query-answering. *SIAM Journal on Computing*, 15(3):703–724, 1986.
- [9] Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal on Computing*, 17(3):427–462, 1988. Preliminary version in FOCS 1985.
- [10] Bernard Chazelle. Lower bounds for orthogonal range searching: I. the reporting case. *J. ACM*, 37(2):200–212, 1990.
- [11] Bernard Chazelle. Lower bounds for orthogonal range searching II. the arithmetic model. *J. ACM*, 37(3):439–463, 1990.
- [12] Bernard Chazelle and Herbert Edelsbrunner. Linear space data structures for two types of range search. *Discrete & Computational Geometry*, 2:113–126, 1987. Preliminary version in SoCG 1986.
- [13] Bernard Chazelle and Leonidas J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1(2):133–162, 1986.
- [14] Bernard Chazelle and Leonidas J. Guibas. Fractional cascading: II. applications. *Algorithmica*, 1(2):163–191, 1986.
- [15] Arash Farzan, J. Ian Munro, and Rajeev Raman. Succinct indices for range queries with applications to orthogonal range maxima. In *Proc. 39th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 327–338, 2012.
- [16] Harold N. Gabow, Jon Louis Bentley, and Robert Endre Tarjan. Scaling and related techniques for geometry problems. In *Proc. 16th Annual ACM Symposium on Theory of Computing (STOC 1984)*, pages 135–143, 1984.
- [17] Pawel Gawrychowski and Patrick K. Nicholson. Optimal encodings for range top-k, k-selection, and min-max. In *Proc. 42nd International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 9134 of *Lecture Notes in Computer Science*, pages 593–604. Springer, 2015.
- [18] Roberto Grossi, Alessio Orlandi, Rajeev Raman, and S. Srinivasa Rao. More haste, less waste: Lowering the redundancy in fully indexable dictionaries. In *Proc. 26th International Symposium on Theoretical Aspects of Computer Science, (STACS)*, pages 517–528, 2009.
- [19] Marek Karpinski and Yakov Nekrich. Space efficient multi-dimensional range reporting. In *Proc. 15th Annual International Conference on Computing and Combinatorics (COCOON)*, pages 215–224, 2009.
- [20] Edward M. McCreight. Priority search trees. *SIAM Journal on Computing*, 14(2):257–276, 1985.
- [21] Gonzalo Navarro, Rajeev Raman, and Srinivasa Rao Satti. Asymptotically optimal encodings for range selection. In *Proc. 34th International Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 29 of *LIPIcs*, pages 291–301. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014.
- [22] Yakov Nekrich. A data structure for multi-dimensional range reporting. In *Proc. 23rd ACM Symposium on Computational Geometry (SoCG)*, pages 344–353, 2007.
- [23] Yakov Nekrich. External memory range reporting on a grid. In *Proc. 18th International Symposium on Algorithms and Computation (ISAAC)*, pages 525–535, 2007.
- [24] Yakov Nekrich. Space efficient dynamic orthogonal range reporting. *Algorithmica*, 49(2):94–108, 2007.
- [25] Yakov Nekrich. Four-dimensional dominance range reporting in linear space. In *Proc. 36th International Symposium on Computational Geometry (SoCG)*, volume 164 of *LIPIcs*, pages 59:1–59:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
- [26] Yakov Nekrich. Four-dimensional dominance range reporting in linear space. *CoRR*, abs/2003.06742, 2020.
- [27] Yakov Nekrich. New data structures for orthogonal range reporting and range minima queries. *CoRR*, abs/2007.11094, 2020.
- [28] Yakov Nekrich and Gonzalo Navarro. Sorted range reporting. In *Proc. 13th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 271–282, 2012.
- [29] Mark H. Overmars. Efficient data structures for range searching on a grid. *J. Algorithms*, 9(2):254–275, 1988.
- [30] Mihai Patrascu. Unifying the landscape of cell-probe lower bounds. *SIAM J. Comput.*, 40(3):827–847, 2011.
- [31] Mihai Patrascu and Mikkel Thorup. Time-space trade-

- offs for predecessor search. In *Proc. 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 232–240, 2006.
- [32] Sairam Subramanian and Sridhar Ramaswamy. The P-range tree: A new data structure for range searching in secondary memory. In *Proc. 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 378–387, 1995.
- [33] Darren Erik Vengroff and Jeffrey Scott Vitter. Efficient 3-d range searching in external memory. In *Proc. 28th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 192–201, 1996.
- [34] Dan E. Willard. On the application of sheared retrieval to orthogonal range queries. In Alok Aggarwal, editor, *Proc. 2nd Annual ACM SIGACT/SIGGRAPH Symposium on Computational Geometry (SoCG)*, pages 80–89. ACM, 1986.