Four-Dimensional Dominance Range Reporting in Linear Space

Yakov Nekrich

Department of Computer Science, Michigan Technological University, Houghton, MI, USA
yakov.nekrich@googlemail.com

---- **Abstract** ----

In this paper we study the four-dimensional dominance range reporting problem and present data structures with linear or almost-linear space usage. Our results can be also used to answer four-dimensional queries that are bounded on five sides. The first data structure presented in this paper uses linear space and answers queries in $O(\log^{1+\varepsilon} n + k \log^{\varepsilon} n)$ time, where k is the number of reported points, n is the number of points in the data structure, and ε is an arbitrarily small positive constant. Our second data structure uses $O(n \log^{\varepsilon} n)$ space and answers queries in $O(\log n + k)$ time.

These are the first data structures for this problem that use linear (resp. $O(n \log^{\varepsilon} n)$) space and answer queries in poly-logarithmic time. For comparison the fastest previously known linear-space or $O(n \log^{\varepsilon} n)$-space data structure supports queries in $O(n^{\varepsilon} + k)$ time (Bentley and Mauer, 1980). Our results can be generalized to $d \geq 4$ dimensions. For example, we can answer d-dimensional dominance range reporting queries in $O(\log \log n (\log n / \log \log n)^{d-3} + k)$ time using $O(n \log^{d-4+\varepsilon} n)$ space. Compared to the fastest previously known result (Chan, 2013), our data structure reduces the space usage by $O(\log n)$ without increasing the query time.

1 Introduction

In the orthogonal range searching problem we keep a set S of multi-dimensional points in a data structure so that for an arbitrary axis-parallel query rectangle Q some information about points in $Q \cap S$ can be computed efficiently. Range searching is one of the most fundamental and widely studied problems in computational geometry. Typically we want to compute some aggregate function on $Q \cap S$ (range aggregate queries), generate the list of points in S (reporting queries) or determine whether $Q \cap S = \emptyset$ (emptiness queries). In this paper we study the complexity of four-dimensional orthogonal range reporting and orthogonal range emptiness queries in the case of dominance queries and in the case when the query range is bounded on five sides. We demonstrate for the first time that in this scenario both queries can be answered in poly-logarithmic time using linear or almost-linear space.

Range trees, introduced by Lueker [23] in 1978 and Bentley [7] in 1980, provide a solution for the range reporting problem in $O(n \log^d n)$ space and $O(\log^d n + k)$ time for any constant dimension d. Henceforth k denotes the number of points in the answer to a reporting query and n denotes the number of points in the data structure. A number of improvements both in time and in space complexity were obtained in the following decades. See e.g., [24, 12, 17, 18, 16, 13, 32, 14, 15, 34, 35, 5, 6, 26, 25, 27, 22, 31, 11, 10] for a selection of previous works on range reporting and related problems. Surveys of previous results can be found in [3, 4, 29]. We say that a range query is f-sided if the query range is bounded on f sides, i.e., a query can be specified with f inequalities; see Fig. 1 on p. 2. Researchers noticed

(a)　　　　　　　　　　(b)　　　　　　　　　　(c)
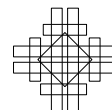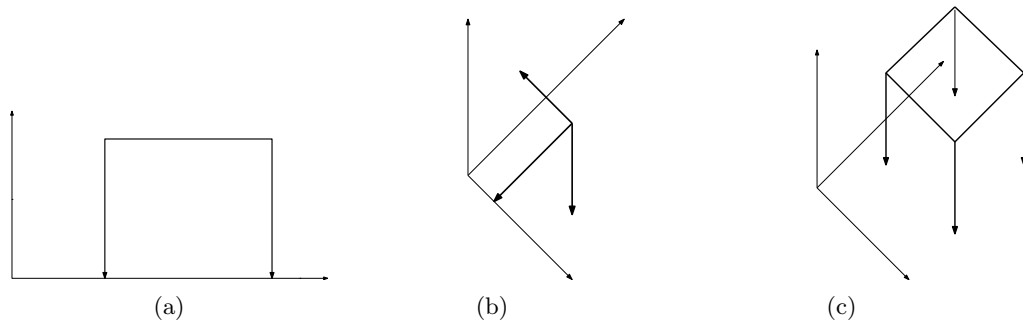
■ **Figure 1** Examples of queries in two and three dimensions. (a) A two-dimensional 3-sided query (b) A three-dimensional 3-sided (dominance) query (c) A three-dimensional 5-sided query.

that the space and time complexity of range reporting depends not only on the dimensionality: the number of sides that bound the query range is also important. Priority search tree, introduced by McCreight [24], provides an $O(n)$ space and $O(\log n + k)$ time solution for 3-sided range reporting queries in two dimensions. In [16] Chazelle and Edelsbrunner have demonstrated that three-dimensional 3-sided queries (aka three-dimensional dominance queries) can be answered in $O(\log^2 n + k)$ time using an $O(n)$ space data structure. In 1985 Chazelle [13] described a compact version of the two-dimensional range tree that uses $O(n)$ space and supports general (4-sided) two-dimensional range reporting queries in $O(\log n + k \log^\varepsilon n)$ time, where ε denotes an arbitrarily small positive constant. In [13] the author also presented an $O(n)$ space data structure that supports 5-sided three-dimensional reporting queries in $O(\log^{1+\varepsilon} n + k \log^\varepsilon n)$ time. Bentley and Mauer [8] described a linear-space data structure that supports d-dimensional range reporting queries for any constant d; however, their data structure has prohibitive query cost $O(n^\varepsilon + k)$.

Summing up, we can answer range reporting queries in poly-logarithmic time using an $O(n)$ space data structure when the query is bounded on at most 5 sides and the query is in two or three dimensions. Significant improvements were achieved on the query complexity of this problem in each case; see Table 1. However, surprisingly, linear-space and polylog-time data structures are known only for the above mentioned special cases of the range reporting. For example, to answer four-dimensional 4-sided queries (four-dimensional dominance queries) in polylogarithmic time using previously known solutions one would need $\Omega(n \log n / \log \log n)$ space. This situation does not change when we increase the space usage to $O(n \log^\varepsilon n)$ words: data structures with poly-logarithmic time are known for the above described special cases only. See Table 2.

Previous results raise the question about low-dimensional range reporting. What determines the complexity of range reporting data structures in $d \geq 4$ dimensions: the dimensionality or the number of sides in the query range? The lower bound of Patrascu [33] resolves this question with respect to query complexity. It is shown in [33] that any data structure using $O(n\texttt{polylog}(n))$ space needs $\Omega(\log n / \log \log n)$ time to answer four-dimensional dominance (4-sided) queries. On the other hand, two- and three-dimensional 4-sided queries can be answered in $O(\log \log n + k)$ time using $O(n\texttt{polylog}(n))$ space. In this paper we address the same question with respect to space complexity.

We demonstrate that four-dimensional 5-sided queries can be answered in $O(\log^{1+\varepsilon} n + k \log^\varepsilon n)$ time using an $O(n)$ space data structure. Our data structure can also support 5-sided emptiness queries in $O(\log^{1+\varepsilon} n)$ time. If the space usage is slightly increased to $O(n \log^\varepsilon n)$, then we can answer reporting and emptiness queries in $O(\log n + k)$ and $O(\log n)$

time respectively. For comparison, the fastest previous method [9] requires $O(n \log^{1+\varepsilon} n)$ space and supports queries in $O(\log n + k)$ time. Since dominance queries are a special case of 5-sided queries, our results can be used to answer four-dimensional dominance queries within the same time and space bounds. Using standard techniques, our results can be generalized to d dimensions for any constant $d \geq 4$: We can answer d-dimensional dominance queries in $O(\log \log n (\log n / \log \log n)^{d-3} + k)$ time and $O(n \log^{d-4+\varepsilon} n)$ space. We can also answer arbitrary $(2d - 3)$-sided d-dimensional range reporting queries within the same time and space bounds.

Our base data structure is the range tree on the fourth coordinate and every tree node contains a data structure that answers three-dimensional dominance queries. We design a space-efficient representation of points stored in tree nodes, so that each point uses only $O(\log \log n)$ bits. Since each point is stored $O(\log n / \log \log n)$ times in our range tree, the total space usage is $O(n \log n)$ bits. Using our representation we can answer three-dimensional queries on tree nodes in $O(\log^\varepsilon n)$ time; we can also "decode" each point, i.e., obtain the actual point coordinates from its compact representation, in $O(\log^\varepsilon n)$ time. Efficient representation of points is the core idea of our method: we keep a geometric construct called *shallow cutting* in each tree node and exploit the relationship between shallow cuttings in different nodes. Shallow cuttings were extensively used in previous works to decrease the query time. But, to the best of our knowledge, this paper is the first that uses shallow cuttings to reduce the space usage. The novel part of our construction is a combination of several shallow cuttings that allows us to navigate between the nodes of the range tree and "decode" points from their compact representations. We recall standard techniques used by our data structure in Section 2. The linear-space data structure is described in Section 3. In Section 4 we show how the decoding cost can be reduced to $O(1)$ by slightly increasing the space usage. The data structure described in Section 4 supports queries in $O(\log n + k)$ time and uses $O(n \log^\varepsilon n)$ space. Previous and new results for 4-sided queries in four dimensions are listed in Table 3. Compared to the only previous linear-space data structure [8], we achieve exponential speed-up in query time. Compared to the fastest previous result [10], our data structure reduces the space usage by $O(\log n)$ without increasing the query time.

The model of computation used in this paper is the standard RAM model. The space is measured in words of $\log n$ bits and we can perform standard arithmetic operations, such as addition and subtraction, in $O(1)$ time. Our data structures rely on bit operations, such as bitwise AND or bit shifts or identifying the most significant bit in a word. However such operations are performed only on small integer values (i.e., on positive integers bounded by n) and can be easily implemented using look-up tables. Thus our data structures can be implemented on the RAM model with standard arithmetic operations and arrays.

We will assume in the rest of this paper that all point coordinates are bounded by n. The general case can be reduced to this special case using the reduction to rank space, described in Section 2. All results of this paper remain valid when the original point coordinates are real numbers.

2 Preliminaries

In this paper ε will denote an arbitrarily small positive constant. We will consider four-dimensional points in a space with coordinate axes denoted by x, y, z, and z'. The following techniques belong to the standard repertoire of geometric data structures.

■ **Table 1** Linear-space data structures for different types of queries. The second column provides the reference to the first data structure achieving linear space and the year of the first publication. The third column contains the query time of the first data structure. The fourth and the fifth columns contain the same information about the best (fastest) currently known data structure. Result marked with E supports only emptiness queries. Data structures in rows 4 and 5 also support 4-sided queries.

Query	First		Best	
Type	Ref	Query Time	Ref	Query Time
2-D 3-sided	[24], 1985	$O(\log n + k)$	[5], 2000	$O(k + 1)$
2-D 4-sided	[13], 1985	$O(\log n + k \log^\varepsilon n)$	[11], 2011	$O((k + 1) \log^\varepsilon n)$
3-D 3-sided	[16], 1986	$O(\log^2 n + k)$	[9], 2011	$O(\log \log n + k)$
3-D 5-sided	[13], 1985	$O(\log^{1+\varepsilon} n + k \log^\varepsilon n)$	[19], 2012	$O(\log n \log \log n)^{\mathrm{E}}$
4-D 5-sided	New	$O(\log^{1+\varepsilon} n + k \log^\varepsilon n)$		

■ **Table 2** $O(n \log^\varepsilon n)$-space data structures for orthogonal range reporting. The second and the third columns contain the reference to and the query time of the first data structure. The fourth and the fifth columns contain the reference to and the query time of the best (fastest) currently known data structure. Data structures in rows 2 and 3 also support 4-sided queries.

Query	First		Best	
Type	Ref	Query Time	Ref	Query Time
2-D 4-sided	[13], 1985	$O(\log n + k)$	[5], 2000	$O(\log \log n + k)$
3-D 5-sided	[13], 1985	$O(\log n + k)$	[11], 2011	$O(\log \log n + k)$
4-D 5-sided	New	$O(\log n + k)$		

Shallow Cuttings. A point q dominates a point p if and only if every coordinate of q is larger than or equal to the corresponding coordinate of p. The *level* of a point q in a set S is the number of points p in S, such that q dominates p (the point q is not necessarily in S). We will say that a *cell* C is a region of space dominated by a point q_c and we will call q_c the apex point of C. A t-shallow cutting of a set S is a collection of cells $\mathcal{C}$, such that (i) every point in $\mathbb{R}^d$ with level at most t (with respect to S) is contained in some cell C_i of $\mathcal{C}$ and (ii) if a point p is contained in some cell C_j of $\mathcal{C}$, then the level of p in S does not exceed $2t$. The size of a shallow cutting is the number of its cells. We can uniquely identify a shallow cutting $\mathcal{C}$ by listing its cells and every cell can be identified by its apex point. Since the level of any point in a cell C_j does not exceed $2t$, every cell C_j contains at most $2t$ points from S, $|C_j \cap S| \leq 2t$ for any C_j in $\mathcal{C}$.

There exists a t-shallow cutting of size $O(n/t)$ for $d = 2$ [35] or $d = 3$ dimensions [1]. Shallow cuttings and related concepts are frequently used in data structures for three-dimensional dominance range reporting queries.

Consider a three-dimensional point $q_3 = (a, b, c)$ and the corresponding dominance range $Q_3 = (-\infty, a] \times (-\infty, b] \times (-\infty, c]$. We can find a cell C of a t-shallow cutting $\mathcal{C}$ that contains q_3 (or report that there is no such C) by answering a point location query in a planar rectangular subdivision of size $O(n/t)$. Point location queries in a rectangular subdivision of size n can be answered in $O(\log \log n)$ time using an $O(n)$-space data structure [10].

Range Trees. A range tree is a data structure that reduces d-dimensional orthogonal range reporting queries to a small number of $(d-1)$-dimensional queries. Range trees provide a general method to solve d-dimensional range reporting queries for any constant dimension d.

Table 3 Previous and new results on dominance range reporting in four dimensions.   Results in lines 2, 3, and 7 can be modified so that space is decreased to $O(n \log n / \log \log n)$ and the query time is increased by $O(\log^\varepsilon n)$ factor.

Ref.	Space	Query Time
[8]	$O(n)$	$O(n^\varepsilon + k)$
[16]	$O(n \log n)$	$O(\log^2 n + k)$
[21]	$O(n \log n)$	$O(\log^2 n / \log \log n + k)$
[25]	$O(n \log^{2+\varepsilon} n)$	$O(\log n \log \log n + k)$
[1]	$O(n \log^{1+\varepsilon} n)$	$O(\log n \log \log n + k)$
[9]	$O(n \log^{1+\varepsilon} n)$	$O(\log n + k)$
[9]	$O(n \log n)$	$O(\log n \log \log n + k)$
New	$O(n \log^\varepsilon n)$	$O(\log n + k)$
New	$O(n)$	$O(\log^{1+\varepsilon} n + k \log^\varepsilon n)$
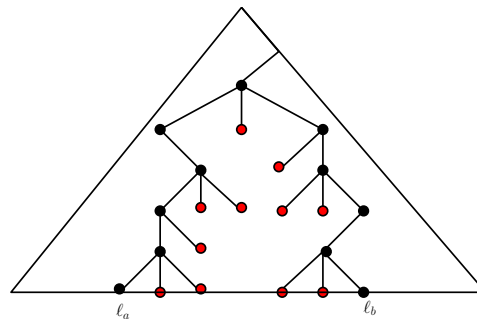
Figure 2 Example of a query in a range tree with node degree $\rho = 3$. Canonical nodes are shown in red. Only nodes of $\pi_a \cup \pi_b$ and $\pi_a' \cup \pi_b'$ are shown.

In this paper we use this data structure to reduce four-dimensional 5-sided reporting queries to three-dimensional 3-sided queries. A range tree for a set S is a balanced tree that holds the points of S in the leaf nodes, sorted by their z'-coordinate. We associated a set $S(u)$ with every internal node u. $S(u)$ contains all points p that are stored in the leaf descendants of u. We assume that each internal node of T has $\rho = \log^\varepsilon n$ children. Thus every point is stored in $O(\log n / \log \log n)$ sets $S(u)$. We keep a data structure that supports three-dimensional reporting queries on $S(u)$ for every node u of the range tree.

Consider a query $Q = Q_3 \times [a, b]$, where Q_3 denotes a 3-sided three-dimensional query range. Let ℓ_a be the rightmost leaf that holds some z'-coordinate $a' < a$ and let ℓ_b be the leftmost leaf that holds some $b' > b$. Let v_{ab} denote the lowest common ancestor of ℓ_a and ℓ_b. We denote by π_a (resp. π_b) the set of nodes on the path from ℓ_a (ℓ_b) to v_{ab}, excluding the node v_{ab}. We will say that u is a left (right) sibling of v iff u and v have the same parent node and u is to the left (respectively, to the right) of v. The set π_a' consists of all nodes u that have some left sibling $v \in \pi_a$ and π_b' consists of all nodes u that have a right sibling $v \in \pi_b$. The set π_a'' (π_b'') consists of all nodes in π_a' (resp. in π_b') that are not children of v_{ab}. The set π_{ab}' consists of all children of v_{ab} that are in $\pi_a' \cap \pi_b'$. For any point $p \in S$, $a \le p.z' \le b$ iff $p \in S(u)$ for some u in $\pi_a'' \cup \pi_b'' \cup \pi_{ab}'$. Nodes $u \in \pi_a'' \cup \pi_b'' \cup \pi_{ab}'$ are called canonical nodes for the range $[a, b]$. See an example on Fig. 2. In order to answer a four-dimensional query Q we visit every canonical node u and report all points $p \in S(u) \cap Q_3$.

Thus we can answer a four-dimensional 5-sided query by answering $O(\rho \cdot \log n / \log \log n)$ three-dimensional 3-sided queries in canonical nodes.

Rank Space.   Let E be a set of numbers. The *rank* of a number x in E is the number of elements in E that do not exceed x: $\mathrm{rank}(x, E) = |\{\, e \in E \,|\, e \leq x \,\}|$. Let $\mathrm{pred}(x, E) = \max\{\, e \in E \,|\, e \leq x \,\}$ and $\mathrm{succ}(x, E) = \min\{\, e \in E \,|\, e \geq x \,\}$. An element $e \in E$ is in the range $[a, b]$, $a \leq e \leq b$, iff its rank satisfies the inequality $a' \leq \mathrm{rank}(e, E) \leq b'$ where $a' = \mathrm{rank}(\mathrm{succ}(a, E), E)$ and $b' = \mathrm{rank}(b, E)$. Hence we can report all $e \in E$ satisfying $a \leq e \leq b$ by finding all elements e satisfying $a' \leq \mathrm{rank}(e, E) \leq b'$.

The same approach can be also extended to multi-dimensional range reporting. A three-dimensional transformation is implemented as follows. We say that a three-dimensional point $p \in S$ is reduced to rank space (or p is in the rank space) if each coordinate of p is replaced by its rank in the set of corresponding coordinates. That is, each point $p = (p.x, p.y, p.z)$ is replaced with $\xi(p) = (\mathrm{rank}(p.x, S_x), \mathrm{rank}(p.y, S.y), \mathrm{rank}(p.z, S_z))$, where S_x, S_y, and S_z denote the sets of x-, y-, and z-coordinates of points in S. For a point $p \in S$ we have

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

where $a' = \mathrm{rank}(\mathrm{succ}(a, S_x), S_x)$, $c' = \mathrm{rank}(\mathrm{succ}(c, S_y), S_y)$, $e' = \mathrm{rank}(\mathrm{succ}(e, S_z), S_z)$, $b' = \mathrm{rank}(b, S_x)$, $d' = \mathrm{rank}(d, S_y)$, and $f' = \mathrm{rank}(f, S_z)$. Thus we can reduce three-dimensional queries on a set S to three-dimensional queries on a set $\{\, \xi(p) \,|\, p \in S \,\}$. Suppose that we store the set $\xi(S)$ in a data structure that answers queries in time $t(n)$ and uses space $s(n)$. Suppose that we also keep data structures that answers predecessor queries on S_x, S_y, and S_z. Then we can answer orthogonal range reporting queries on S in time $t(n) + O(t_{\mathrm{pred}}(n))$ using space $s(n) + O(s_{\mathrm{pred}}(n))$. Here $s_{\mathrm{pred}}(n)$ and $t_{\mathrm{pred}}(n)$ are the space usage and query time of the predecessor data structure. Additionally we need a look-up table that computes $\xi^{-1}(p)$, i.e., computes the coordinates of a point p from its coordinates in the rank space. As we will show later, in some situations this table is not necessary. Summing up, reduction to rank space enables us to reduce the range reporting problem on a set $S \subset \mathbb{R}^3$ to a special case when all point coordinates are positive integers bounded by $|S|$.

The same rank reduction technique can be applied to range reporting in any constant dimension d. In the rest of this paper we will assume for simplicity that all points of S are in the rank space.

3   Five-Sided Range Reporting in Linear Space

Base Structure.   We keep all points in a range tree that is built on the fourth coordinate. Each tree node has $\rho = \log^\varepsilon n$ children; thus the tree height is $O(\log n / \log \log n)$. Let $S(u)$ denote the set of points assigned to a node u. To simplify the notation, we will not distinguish between points in $S(u)$ and their projections onto (x, y, z)-space. We will say for example that a point p is in a range $Q = (-\infty, a] \times (-\infty, b] \times (-\infty, c]$ if the projection of p onto (x, y, z)-space is in Q.

Since we aim for a linear-space data structure, we cannot store sets $S(u)$ in the nodes of the range tree. We keep a t_0-shallow cutting $\mathcal{C}(u)$ of $S(u)$ where $t_0 = \log^2 n$. For every cell $C_i(u)$ of the shallow cutting we store all points from $S(u) \cap C_i$ in a data structure supporting three-dimensional dominance queries. We do not store the original (real) coordinates of points[1] in C_i. Instead we keep coordinates in the rank space of $S(u) \cap C_i$. Since $S(u) \cap C_i$ contains $O(\log^2 n)$ points, we need only $O(\log \log n)$ bits per point to answer three-dimensional dominance queries in the rank space.

---

[1] To avoid clumsy notation, we will sometimes omit the node specification when the node is clear from the context. Thus we will sometimes write C_i instead of $C_i(u)$ and $\mathcal{C}$ instead of $\mathcal{C}(u)$. The same simplification will be used for other shallow cuttings.

We can answer a 5-sided query $(-\infty, q_x] \times (-\infty, q_y] \times (-\infty, q_z] \times [z_l', z_r']$ by visiting all canonical nodes that cover the range $[z_l', z_r']$. In every visited node we answer a three-dimensional dominance query, i.e., report all points dominated by $q_3 = (q_x, q_y, q_z)$ in two steps: first, we search for some cell $C_i(u)$ that contains q_3. If such a cell $C_i(u)$ exists, then we answer the dominance query in the rank space of $S(u) \cap C_i(u)$ in $O(1)$ time per point. We describe the data structure for dominance queries on t_0 rank-reduced points in the full version of this paper [30].

We must address several issues in order to obtain a working solution: How can we transform a three-dimensional query to the rank space of $C_i(u)$? A data structure for cell $C_i(u)$ reports points in the rank space of $C_i(u)$. How can we obtain the original point coordinates? Finally how do we answer a query on $S(u)$ if q_3 is not contained in any cell $C_i(u)$? First, we will explain how to decode points from a cell $C(u)$ and obtain their original coordinates. We also show how to transform a query to the rank space of a cell. Next we will describe a complete procedure for answering a query. Finally we will improve the query time and achieve the main result of this section.

Decoding Points. This is the crucial component of our construction. We will need additional structures to obtain the original coordinates of points from $\mathcal{C}(u)$. To this end we keep an additional $(4t_0)$-shallow cutting $\mathcal{C}'(u)$ in every node of the range tree. For each cell $C_i'(u)$ of $\mathcal{C}'(u)$ we create a separate $(2t_0)$-shallow cutting of $S(u) \cap C_i'(u)$, called $\mathcal{D}_i(u)$.

▶ **Lemma 1.** *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 $\mathcal{B}$.*

Proof. Consider an apex point a_i of A_i (i.e., the point with maximum x- y-, and z-coordinates in A_i). The level of a_i in S is at most $2f$ by definition of a shallow cutting. Since $S' \subseteq S$, the level of a_i in S' does not exceed $2f$. Hence there exists a cell B_j of $\mathcal{B}$ that contains a_i. The apex b_j of B_j dominates a_i. Hence b_j also dominates all points from A_i and B_j contains A_i. ◀

Lemma 1 will be extensively used in our decoding procedure. We will say that a point p in $C_i'(u)$ is *interesting* if p is contained in some $C_k(w)$, where w is an ancestor of u. Each interesting point $p \in S(u)$ can be uniquely represented by (a) a cell $C_i'(u)$ of $\mathcal{C}'(u)$ that contains p and (b) coordinates of p in the rank space of $C_i'(u)$. The following relationship between shallow cuttings provides the key insight.

▶ **Lemma 2.**
 (i) *Every cell $C_i(u)$ of $\mathcal{C}(u)$ is contained in some cell $C_j'(u)$ of $\mathcal{C}'(u)$*
 (ii) *Let u_r be a child of an internal node u. Every cell $D_{ij}(u)$ of every $\mathcal{D}_i(u)$ is contained in some cell $C_k'(u_r)$ of $\mathcal{C}'(u_r)$.*
 (iii) *Every interesting point from $C_i'(u)$ is stored in some cell $D_{ij}(u)$ of $\mathcal{D}_i(u)$.*

Proof. (i) Immediately follows from Lemma 1. (ii) Consider the apex point p_a of $D_{ij}(u)$. The point p_a dominates at most $4t_0$ points from $S(u)$ and at most $4t_0$ points from $S(u_r)$ because $S(u_r) \subset S(u)$. Hence both p_a and $D_{ij}(u)$ are contained in some cell of $\mathcal{C}'(u_r)$. (iii) Suppose that a point $p \in S(w)$ is stored in some cell $C_k(w)$ of $\mathcal{C}(w)$ where w is an ancestor of u. The point p dominates at most $2t_0$ points from $S(w)$. Since $S(u)$ is a subset of $S(w)$, p dominates at most $2t_0$ points in $S(u)$. Hence p is contained in some cell $C_i'(u)$ of the shallow cutting $\mathcal{C}'(u)$. Every point of $C_i'(u) \cap S(u)$ that dominates at most $2t_0$ points of $S(u)$ is contained in some cell $D_{ij}(u)$ of $\mathcal{D}_i(u)$. ◀

Consider an arbitrary point p in a cell $C_i(u)$ of $\mathcal{C}(u)$. Our decoding procedure finds a representation of p in $\mathcal{C}'(u)$. That is, we find the cell $C'_j(u)$ of $\mathcal{C}'(u)$, such that $p \in C'_j(u)$, and the rank of p in $C'_j(u)$. The key observation is that $C_i(u)$ is contained in some $C'_j(u)$ (Lemma 2, item (i)) Therefore we need to store a pointer to $C'_j(u)$ only once for all points p in $C_i(u)$. For every p from $C_i(u)$, we can store its rank in $C'_j(u)$ using $O(\log \log n)$ bits. Next, our decoding procedure moves from a node u to its child u_f, such that $p \in S(u_f)$, and computes a representation of p in $\mathcal{C}'(u_f)$. This is done in two steps: First we examine the shallow cutting $\mathcal{D}_j(u)$ and find the cell $D_{jl}(u)$ that contains p. By Lemma 2, item (iii), such a cell always exists. The shallow cutting $\mathcal{D}_{jl}(u)$ consists of $O(1)$ cells. Therefore we can store, for any interesting point p, the cell $D_{jl}(u)$ containing p and the rank of p in $D_{jl}(u)$ using $O(\log \log n)$ bits. Then we move from $D_{jl}(u)$ to $\mathcal{C}(u_f)$: by Lemma 2, item (ii), $D_{jl}(u)$ is contained in some $C'_k(u_f)$. Thus we need to store the pointer to $C'_k(u_f)$ only once for all interesting points p in $D_{jl}(u)$. We can store the rank of p in $C'_k(u_f)$ using $O(\log \log n)$ bits. When the representation of p in $\mathcal{C}'(u_f)$ is known, we move to the child u_d of u_f, such that $p \in S(u_d)$ and compute a representation of p in $\mathcal{C}'(u_d)$. We continue in the same way until a leaf node is reached. Every leaf node ℓ contains original (real) coordinates of points in $S(\ell)$. Hence we can obtain the coordinates of p when a leaf is reached. Summing up, shallow cuttings $\mathcal{C}'(u)$ and $\mathcal{D}_i(u)$ allow us to move from a node u to a child of u using only $O(\log \log n)$ additional bits per point. A more detailed description of auxiliary data structures needed for decoding is given in the next paragraph.

For each cell $C_i(u)$ of $\mathcal{C}(u)$ we keep a pointer to the cell $C'_{\mathtt{cont}(i)}(u)$ of $\mathcal{C}'(u)$ that contains $C_i(u)$. For every cell $D_{ij}(u) \in \mathcal{D}_i(u)$ and for each child u_r of u, we keep a pointer to the cell $C'_{\mathtt{down}(i,j,r)}(u_r) \in \mathcal{C}'(u_r)$, such that $C'_{\mathtt{down}(i,j,r)}(u_r)$ contains $D_{ij}(u)$. We can identify a point in each cell of a shallow cutting $\mathcal{C}(u)$ (resp. $\mathcal{C}'(u)$ or $\mathcal{D}'_i(u)$) with $O(\log \log n)$ bits because each cell contains a poly-logarithmic number of points. The x-rank of a point in a cell will be used as its identifier. We keep a mapping from points in a cell C_i to the corresponding points in a containing cell $C'_{\mathtt{cont}(i)}$. The array $F_X(C_i)$ maps x-ranks of points in C_i to their x-ranks in $C'_{\mathtt{cont}(i)}$: if the x-rank of a point $p \in C_i$ is equal to f, then $F_X[f] = g$ where g is the x-rank of p in $C'_{\mathtt{cont}(i)}$. The array $F''_{X,r}$ for a cell $D_{ij}(u)$ and a child u_r of u maps x-ranks of points in $D_{ij}(u)$ to their x-ranks in $C'_{\mathtt{down}(i,j,r)}$. If the x-rank of a point $p \in C_i$ is equal to f, then $F''_{X,r}[f] = g$ where g is the x-rank of p in $C'_{\mathtt{down}(i,j,r)}$. We also keep a mapping from $C'_i(u)$ to cells of $\mathcal{D}_i(u)$: for every point $p \in C'_i(u)$ we store the cell D_{ij} that contains p and the x-rank of p in C_{ij} (or $NULL$ if p is not in $\mathcal{C}_{ij}$). For every point p in each cell $D_{ij}(u)$ of $\mathcal{C}'_i(u)$, we store the index of the child u_r such that $p \in S(u_r)$. Our method requires $O(\log \log n)$ bits per point. Each pointer from $C_i(u)$ to $C'_{\mathtt{cont}(i)}(u)$ and from $D_{ij}(u)$ to $C'_{\mathtt{down}(i,j,r)}(u)$ consumes $O(\log n)$ bits. We store $O(\log^{2\varepsilon} n)$ pointers per cell and there are $O(n/(\log n \log \log n))$ cells in all shallow cuttings of the range tree. Hence the total space used by all pointers is $O(n \log^{2\varepsilon} n)$ bits.

▶ **Lemma 3.** *For any interesting point p in a cell $C'_i(u)$, we can find the representation of p in $C'_i(u_f)$, where u_f is the child of u that contains p.*

Proof. First we identify the cell $D_{ij}(u)$ of $\mathcal{D}_i(u)$ that contains p and compute the x-rank of p in $D_{ij}(u)$. Since p is interesting, such a cell exists. Then we use the array $F''_{X,k}$ of this cell and find the x-rank of p in the cell $C'_{\mathtt{down}(i,j,k)}$. ◀

For any point from $C_i(u)$ we can obtain its position in some cell $C'_{\mathtt{cont}(i)}(u)$ in $O(1)$ time. Then we can move down and obtain its representation in a child of u in $O(1)$ time. We can access the original coordinates of p when a leaf node is reached. Thus we can "decode" a point p if we know its position in a cell $C_i(u)$ in $O(\log n / \log \log n)$ time.

We can reduce a three-dimensional query $(-\infty, a] \times (-\infty, b] \times (-\infty, c]$ to the rank space of a cell by binary search. Let $X(C_i)$ denote the list of points in a cell C_i sorted by x-coordinates. To compare a with the x-coordinate of $X(C_i)[g]$ for some index g, we decode the point $p = X(C_i)[g]$ as explained above. Hence we can find the predecessor of a in $X(C_i)$ by binary search in $O(\log \log n)$ time. We can find the predecessor of b in $Y(C_i)$ and the predecessor of c in $Z(C_i)$ using the same procedure, where $Y(C_i)$ and $Z(C_i)$ are the lists of points in C_i sorted by their y- and z-coordinates respectively.

Queries. Consider a four-dimensional 5-sided query $(-\infty, a] \times (-\infty, b] \times (-\infty, c] \times [z'_l, z'_r]$. We visit all canonical nodes that cover the range $[z'_l, z'_r]$. In every visited node we answer a three-dimensional query using the following procedure. We find a cell $C_i(u)$ that contains p. We transform $(-\infty, a] \times (-\infty, b] \times (-\infty, c]$ to the rank space of $C_i(u)$ and answer the transformed query on $C_i(u) \cap S(u)$. Every reported point is decoded using the procedure described above. If there is no cell $C_i(u)$ that contains p, then p dominates at least t_0 points from $S(u)$. In this case we visit all children of u and recursively answer three-dimensional dominance query in each child using the same procedure.

We need $O(\log \log n)$ time to find the cell $C_i(u)$ or determine that $C_i(u)$ does not exist. To answer a query on $C_i(u)$ we need $O(\log n)$ time (ignoring the time to report points, but taking into account the time that we need to transform a query to the rank space of $C_i(u)$). Thus the total time spent in a node u is $O(\log n)$. The time spent in descendants of u can be estimated as follows. Let T_u be the subtree of the range tree induced by u and its visited descendants. Let T'_u denote the subtree of T_u obtained by removing all leaves of T_u. Every leaf of T'_u is an internal node of T_u. Hence we report at least t_0 points for every leaf in T'_u. The height of T'_u is bounded by $O(\log n / \log \log n)$. Let l_u denote the number of leaves in T'_u. The total number of nodes in T'_u is bounded by $O(l_u \log n / \log \log n)$. Every node of T'_u has ρ children. Hence the total number of nodes in T_u does not exceed $O(l_u (\log^{1+\varepsilon} n / \log \log n))$. The time spent in all nodes of T_u can be bounded by $O(l_u \log^{2+\varepsilon} n)$ (again, ignoring the time to decode and report points). When we visit descendants of u we report at least $k_u = \Omega(l_u \cdot t_0)$ points and each point is decoded in $O(\log n / \log \log n)$ time. The total time spent in descendants of u is $O(l_u \log^{2+\varepsilon} n + k_u (\log n / \log \log n)) = O(k_u (\log n / \log \log n))$. The time spent in all canonical nodes and their descendants is $O(\log^{2+\varepsilon} n + k (\log n / \log \log n))$.

Faster Decoding. We can speed-up the decoding procedure and thus the overall query time without increasing the asymptotic space usage. Our approach is very similar to the method used in compact two-dimensional range trees [13, 28, 11]. All nodes in the range tree are classified according to their depth. A node u is an i-node if the depth h_u of u divides ρ^i where $\rho = \log^\varepsilon n$, $h_u = x \cdot \rho^i$ for some $i \geq 0$, but h_u does not divide ρ^j for $j > i$. We keep an additional $4t_i$-shallow cutting $\mathcal{C}^i$ in every i-node u where $t_i = \rho^i \cdot \log^2 n$. As before for each cell C_i^j of $\mathcal{C}^j$ we construct a $2t_j$-shallow cutting $\mathcal{D}_i^j$. Let an i-descendant of a node u denote the highest i-node v that is a descendant of u. If a node u is an i-node, then it has ρ^i i-descendants. For every cell D_{ik}^j of each $\mathcal{D}_i^j$ and for every j-descendant u_l of u, we keep the index $r = \mathtt{down}(j, i, l)$ of the cell $C_r^j(u_l)$ that contains $D_{ik}^j(u)$. For each point in $D_{ik}^j(u)$ we keep the index l of the i-descendant that contains p and the x-rank of p in $C_r^j(u_l)$ where $r = \mathtt{down}(j, i, l)$.

Using these additional shallow cuttings, we can reduce the decoding time to $O(\log^\varepsilon n)$. To decode a point p in $S(u)$ we move down from a node u to its child $u_{0,1}$ and find a representation of p in $\mathcal{C}'(u_{0,1})$. Then we move to the child $u_{0,2}$ of $u_{0,1}$ and continue in the same manner until a 1-node $u_{1,1}$ is reached. Next we move from $u_{1,1}$ to its 1-descendant $u_{1,2}$,

then to a 1-descendant $u_{1,3}$ of $u_{1,2}$, and so on until a 2-node is reached. During the j-th iteration we move down along a sequence of j-nodes until a $(j+1)$-node or a leaf node is reached. During each iteration we visit $O(\log^\varepsilon n)$ nodes and spend $O(1)$ time in every node. There are at most $\log_\rho \log n = O(1/\varepsilon)$ iterations. Hence the decoding time for a point is $O(\log^\varepsilon n)$. The total query time is reduced to $O(\log^{1+2\varepsilon} n + k \log^\varepsilon n)$. If we replace ε with $\varepsilon/2$ in the above proof, we obtain our first result.

▶ **Theorem 4.** *There exists a linear-space data structure that answers four-dimensional 5-sided reporting queries in $O(\log^{1+\varepsilon} n + k \log^\varepsilon n)$ time and four-dimensional 5-sided emptiness queries in $O(\log^{1+\varepsilon} n)$ time.*

4   Faster Queries using More Space

In this section we will show how to reduce the decoding time to $O(1)$ per point by increasing the space usage. We make several modifications in the basic construction of Section 3.

For any i and j such that $1 \leq i \leq j \leq \rho$ and for any internal node u of the range tree, we store the set $S(u, i, j)$. $S(u, i, j)$ is the union of sets $S(u_i)$, $S(u_{i+1})$, ..., $S(u_j)$. For every set $S(u, i, j)$ we construct a t_0-shallow cutting $\mathcal{C}(u, i, j)$. For each cell C_l of $\mathcal{C}(u, i, j)$ we store a three-dimensional data structure that keeps points from $C_l \cap S(u, i, j)$ in the rank space and answers three-dimensional dominance queries in $O(k+1)$ time.

The decoding procedure is implemented in the same way as in Section 3, but with different parameter values. Recall that a node u is an i-node for some $i \geq 0$ if the depth of u divides ρ^i but does not divide ρ^{i+1}. We keep an additional $4t_{i+1}$-shallow cutting $\mathcal{C}^i(u, l, r)$ for every i-node u and every pair $1 \leq l \leq r \leq \rho$. For every cell C_s of $\mathcal{C}^i(u, l, r)$ we keep a $2t_{i+1}$-shallow cutting $\mathcal{D}_s$. Consider a cell D_g of $\mathcal{D}_s$. For every $(i+1)$-descendant v of u and for every pair l, r satisfying $1 \leq l \leq r \leq \rho$, we keep the index $x = \mathtt{down}(D_g, v, l, r)$ such that the cell C_x of $\mathcal{C}^{i+1}(v, l, r)$ contains D_g. We also store a mapping from $\mathcal{C}(u, l, r)$ to $\mathcal{C}^i(u, l, r)$ for every i-node u. That is, for every cell C_f of $\mathcal{C}(u, l, r)$ we keep the index $g = \mathtt{cont}(f)$, such that the cell $C_g \in \mathcal{C}^i(u, l, r)$ contains C_f; for every point $p \in S(u) \cap C_f$ we keep its identifier in $C_{\mathtt{cont}(f)}$. For every cell C_g of $\mathcal{C}^i(u, l, r)$ we keep a mapping from C_g to $\mathcal{D}_g$. That is, for every point p in $C_g \cap S(u, l, r)$ we store the cell D_s of $\mathcal{D}_g$ that contains p and the identifier of p in D_s. Finally we also store a mapping from every cell D_s of each $\mathcal{D}_g$ to shallow cuttings in $(i+1)$-descendants of u. For every point $p \in D_s \cap S(u, l, r)$ we store (i) the i-descendant v of u such that $p \in S(v)$ and (ii) the identifier of p in C_x where $x = \mathtt{down}(D_s, v, l, r)$.

Our modified data structure uses $O(n \log^{3\varepsilon} n)$ words of space. The representation of a point in $\mathcal{C}(u, i, j)$ takes $O(\log \log n)$ bits per point and every point is stored in $O(\log^{1+2\varepsilon} n / \log \log n)$ shallow cuttings $\mathcal{C}(u, i, j)$. The mapping from $\mathcal{C}(u, l, r)$ to $\mathcal{C}^i(u, l, r)$ in an i-node u takes $O(\log^{(i+1)\varepsilon} n)$ bits per point. We also need $O(\log^{(i+1)\varepsilon} n)$ bits per point to store the mapping from a cell C_g of $\mathcal{C}^i(u, l, r)$ to $\mathcal{D}_g$. The mapping from a cell D_s of $\mathcal{D}_g$ to shallow cuttings in $(i+1)$-descendants of u consumes the same space. The total number of points in all $S(u)$ where u is an i-node is $O(n \log^{1-i\varepsilon} n)$. The total number of points in all $S(u, l, r)$ where u is an i-node and $1 \leq l \leq r \leq \rho$ is $O(n \log^{1+(2-i)\varepsilon} n)$. Hence the total space used by all mappings in all i-nodes is $O(n \log^{1+3\varepsilon} n)$ bits or $O(n \log^{3\varepsilon} n)$ words of $\log n$ bits.

Every point p in $C_i \cap S(u, l, r)$, where C_i is a cell of $\mathcal{C}(u, l, r)$, can be decoded in $O(1)$ time. Suppose that u is a j-node. Using the mapping from C_i to $\mathcal{C}^j(u, l, r)$, we can find the representation of p in $\mathcal{C}^j(u, l, r)$, i.e., a cell C_s that contains p and the identifier of p in C_s. If we know the identifier of p in C_s, we can find the representation of p in $\mathcal{D}_s$. Using the mapping from a cell of $\mathcal{D}_s$ to $(j+1)$-descendants of u, we can compute the representation of p in a cell C_v of $\mathcal{C}^{j+1}(v, l', r')$, where v is a direct $(j+1)$-descendant of u. Thus we moved

from a j-node to its $(j + 1)$-descendant in $O(1)$ time. We continue in the same way and move to a $(j + 2)$-descendant of u, then a $(j + 3)$-descendant of u, and so on. After at most $(1/\varepsilon) = O(1)$ iterations, we reach a leaf node and obtain the original coordinates of p.

We can translate a query $(-\infty, a] \times (-\infty, b] \times (-\infty, c]$ into the rank space of a cell C_i in constant time. Let $X(C_i)$ denote the list of x-coordinates of $S(u, l, r) \cap C_i$. We keep $X(C_i)$ in the compact trie data structure of [20]. This data structure requires $O(\log \log n)$ bits per point. Elements of $X(C_i)$ are not stored in the compact trie; we only store some auxiliary information using $O(\log \log n)$ bits per element. Compact trie supports predecessor queries on $X(C_i)$ in $O(1)$ time, but the search procedure must access $O(1)$ elements of $X(C_i)$. Since we can decode a point from C_i in $O(1)$ time, we can also access an element of $X(C_i)$ in $O(1)$ time. Hence, we can compute the predecessor of a in $X(C_i)$ (and its rank) in $O(1)$ time. We can translate b and c to the rank space in the same way.

Queries. Consider a four-dimensional 5-sided query $(-\infty, a] \times (-\infty, b] \times (-\infty, c] \times [z'_1, z'_2]$. There are $O(\log n / \log \log n)$ canonical sets $S(u_i, l_i, r_i)$, such that $p.z \in [z'_1, z'_2]$ iff $p \in S(u_i, l_i, r_i)$ for some i. Canonical sets can be found as follows. Let ℓ_1 be the leaf that holds the largest $l_1 < z'_1$ and ℓ_2 be the leaf that holds the smallest $l_2 > z'_2$. Let v denote the lowest common ancestor of ℓ_1 and ℓ_2. Let π_1 denote the path from ℓ_1 to v (excluding v) and let π_2 denote the path from ℓ_2 to v (excluding v). For each node $u \in \pi_2$, we consider a canonical set $S(u, l, r)$ such that $u_l, \ldots, u_r$ are left siblings of some node $u_{r+1} \in \pi_2$. For each node $u \in \pi_1$, we consider a canonical set $S(u, l, r)$ such that $u_l, \ldots, u_r$ are right siblings of some node $u_{l-1} \in \pi_1$. Finally we consider the set $S(v, l, r)$ such that $v_l, \ldots, v_r$ have a left sibling on π_1 and a right sibling on π_2. The fourth coordinate of a point p is in the interval $[z'_1, z'_2]$ iff p is stored in one of the canonical sets described above. Hence we need to visit all canonical sets and answer a three-dimensional query $(-\infty, a] \times (-\infty, b] \times (-\infty, c]$ in each set.

There are $O(\log n / \log \log n)$ canonical sets $S(u, l, r)$. Each canonical set is processed as follows. We find the cell C_u of $\mathcal{C}(u, l, r)$ that contains $q_3 = (a, b, c)$. Then we translate q_3 into the rank space of $C_u \cap S(u, l, r)$ and answer the dominance query. Reported points are decoded in $O(1)$ time per point as explained above. We can also translate the query into the rank space of C_u in $O(1)$ time. If q_3 is not contained in any cell of $\mathcal{C}(u, l, r)$, then q_3 dominates at least $\log^2 n$ points of $S(u, l, r)$. We visit all children u_i of u for $l \le i \le r$ and recursively answer the dominance query in each child. Using the same arguments as in Section 3, we can show that the total number of visited nodes does not exceed $O(k / \log^\varepsilon n)$, where k is the number of reported points.

If we replace ε with $\varepsilon/3$ in the above proof, we obtain the following result.

▶ **Theorem 5.** *There exists an $O(n \log^\varepsilon n)$ space data structure that answers four-dimensional 5-sided reporting queries in $O(\log n + k)$ time and four-dimensional 5-sided emptiness queries in $O(\log n)$ time.*

We can extend our result to support dominance queries (or any $(2d - 3)$-sided queries) in $d \ge 4$ dimensions using standard techniques.

▶ **Theorem 6.** *There exists an $O(n \log^{d-4+\varepsilon} n)$ space data structure that supports d-dimensional dominance range reporting queries in $O(\log^{d-3} n / (\log \log n)^{d-4} + k)$ time for any constant $d \ge 4$.*
There exists an $O(n \log^{d-4+\varepsilon} n)$ space data structure that supports d-dimensional $(2d-3)$-sided range reporting queries in $O(\log^{d-3} n / (\log \log n)^{d-4} + k)$ time for any constant $d \ge 4$.

5   Conclusions

In this paper we described data structures with linear and almost-linear space usage that answer four-dimensional range reporting queries in poly-logarithmic time provided that the query range is bounded on 5 sides. This scenario includes an important special case of dominance range reporting queries that was studied in a number of previous works [16, 35, 25, 1, 11, 10]; for instance, the offline variant of four-dimensional dominance reporting is used to solve the rectangle enclosure problem [11, 2]. Our result immediately leads to better data structures in $d \geq 4$ dimensions. E.g., we can answer d-dimensional dominance range reporting queries in $O(n \log^{d-4+\varepsilon} n)$ space and $O(\log \log n (\log n / \log \log n)^{d-3})$ time. We expect that the methods of this paper can be applied to other geometric problems, such as the offline rectangle enclosure problem.

Our result demonstrates that the space complexity of four-dimensional queries is determined by the number of sides, i.e., the number of inequalities that are needed to specify the query range. This raises the question about the space complexity of dominance range reporting in five dimensions. Is it possible to construct a linear-space data structure that supports five-dimensional dominance range reporting queries in poly-logarithmic time?

Compared to the fastest previous solution for the four-dimensional dominance range reporting problem [10], our method decreases the space usage by $O(\log n)$ factor without increasing the query time. However, there is still a small gap between the $O(\log n + k)$ query time, achieved by the fastest data structures, and the lower bound of $\Omega(\log n / \log \log n)$, proved in [33]. Closing this gap is another interesting open problem.

—— **References** ——

1   Peyman Afshani. On dominance reporting in 3d. In *Proc. 16th Annual European Symposium on Algorithms (ESA)*, pages 41–51, 2008.

2   Peyman Afshani, Timothy M. Chan, and Konstantinos Tsakalidis. Deterministic rectangle enclosure and offline dominance reporting on the RAM. In *Proceedings of 41st International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 77–88, 2014. `doi:10.1007/978-3-662-43948-7_7`.

3   Pankaj K. Agarwal. Range searching. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry, Second Edition.*, pages 809–837. Chapman and Hall/CRC, 2004. `doi:10.1201/9781420035315.ch36`.

4   Pankaj K. Agarwal and Jeff Erickson. Geometric range searching and its relatives. *Contemporary Mathematics*, 223:1–56, 1999.

5   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. `doi:10.1109/SFCS.2000.892088`.

6   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. `doi:10.1145/380752.380842`.

7   Jon Louis Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229, 1980. `doi:10.1145/358841.358850`.

8   Jon Louis Bentley and Hermann A. Maurer. Efficient worst-case data structures for range searching. *Acta Inf.*, 13:155–168, 1980. `doi:10.1007/BF00263991`.

9   Timothy M. Chan. Persistent predecessor search and orthogonal point location on the word RAM. In *Proc. 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1131–1145, 2011. `doi:10.1137/1.9781611973082.85`.

10 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. `doi:10.1145/2483699.2483702`.

11 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.

12 Bernard Chazelle. Filtering search: a new approach to query-answering. *SIAM Journal on Computing*, 15(3):703–724, 1986. `doi:10.1137/0215051`.

13 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.

14 Bernard Chazelle. Lower bounds for orthogonal range searching: I. the reporting case. *J. ACM*, 37(2):200–212, 1990. `doi:10.1145/77600.77614`.

15 Bernard Chazelle. Lower bounds for orthogonal range searching II. the arithmetic model. *J. ACM*, 37(3):439–463, 1990. `doi:10.1145/79147.79149`.

16 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. `doi:10.1007/BF02187875`.

17 Bernard Chazelle and Leonidas J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1(2):133–162, 1986. `doi:10.1007/BF01840440`.

18 Bernard Chazelle and Leonidas J. Guibas. Fractional cascading: II. applications. *Algorithmica*, 1(2):163–191, 1986. `doi:10.1007/BF01840441`.

19 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. `doi:10.1007/978-3-642-31594-7_28`.

20 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. `doi:10.4230/LIPIcs.STACS.2009.1847`.

21 Joseph JáJá, Christian Worm Mortensen, and Qingmin Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In *Proc. 15th International Symposium on Algorithms and Computation (ISAAC)*, pages 558–568, 2004. `doi:10.1007/978-3-540-30551-4_49`.

22 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. `doi:10.1007/978-3-642-02882-3_22`.

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

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

25 Yakov Nekrich. A data structure for multi-dimensional range reporting. In *Proc. 23rd ACM Symposium on Computational Geometry (SoCG)*, pages 344–353, 2007. `doi:10.1145/1247069.1247130`.

26 Yakov Nekrich. External memory range reporting on a grid. In *Proc. 18th International Symposium on Algorithms and Computation (ISAAC)*, pages 525–535, 2007. `doi:10.1007/978-3-540-77120-3_46`.

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

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

29   Yakov Nekrich. Orthogonal range searching on discrete grids. In *Encyclopedia of Algorithms*, pages 1484–1489. Springer, New York, 2016. `doi:10.1007/978-1-4939-2864-4_631`.

30   Yakov Nekrich. Four-dimensional dominance range reporting in linear space. *CoRR*, abs/2003.06742, 2020. `arXiv:2003.06742`.

31   Yakov Nekrich and Gonzalo Navarro. Sorted range reporting. In *Proc. 13th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 271–282, 2012. `doi:10.1007/978-3-642-31155-0_24`.

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

33   Mihai Patrascu. Unifying the landscape of cell-probe lower bounds. *SIAM J. Comput.*, 40(3):827–847, 2011. `doi:10.1137/09075336X`.

34   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.

35   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.