Orthogonal Range Reporting in Three and Higher Dimensions

Peyman Afshani* Lars Arge* Kasper Dalgaard Larsen*

MADALGO†
Department of Computer Science
Aarhus University, Denmark.

{peyman,large,larsen}@madalgo.au.dk

Abstract

In orthogonal range reporting we are to preprocess N points in d-dimensional space so that the points inside a d-dimensional axis-aligned query box can be reported efficiently. This is a fundamental problem in various fields, including spatial databases and computational geometry.

In this paper we provide a number of improvements for three and higher dimensional orthogonal range reporting: In the pointer machine model, we improve all the best previous results, some of which have not seen any improvements in almost two decades. In the I/O-model, we improve the previously known three-dimensional structures and provide the first (non-trivial) structures for four and higher dimensions.

Keywords-data structures; computational geometry; orthogonal range searching; external memory;

1. Introduction

Orthogonal range reporting is a fundamental problem in several fields, including spatial databases and computational geometry (e.g., see the surveys by Gaede and Günter [21], Arge [6], Agarwal [2] or Agarwal and Erickson [3] for a data base, external memory and computational geometry point of view, respectively). In this problem, we are to preprocess a set of N points in d-dimensional space so that the points in a d-dimensional axis-parallel query box can be reported efficiently. This problem has been studied extensively by many researchers and in different models of computation, including in the pointer machine (e.g., [9], [17], [24]), the RAM (e.g., [5], [12], [20], [25]), and the external memory (e.g., [1], [7], [27], [29]) models. Various lower bounds have also been obtained (e.g., [13], [14], [19], [26]). The problem is especially well-understood in two dimensions, where space and query optimal structures have been developed in most models of computation. This is not the case in higher dimensions.

In this paper we provide a number of improvements for the three and higher dimensional versions of the problem in the pointer machine model and the external memory (I/O) model.

1.1. Previous Pointer Machine Model Results

In this section we review previous pointer machine model orthogonal range reporting results. For brevity, we focus on static near-linear space structures with polylogarithmic query bounds. For other variants see [2], [3].

The one-dimensional version of the problem can be solved with optimal $O(\log N + K)$ query time and linear space using a binary search tree. Here K denotes the number of elements returned by a query. Using priority search trees [24] and fractional cascading [16], Chazelle described a two-dimensional structure using $O(N \log N / \log\log N)$ space that answers queries in $O(\log N + K)$ query time [11]. This is optimal [13]. Three results are known in three and higher dimensions, each offering a different trade-off between query time and space. The first data structure was given by Chazelle and uses $O(N \log^{d-1} / \log\log N)$ space and can answer queries in $O(\log^{d-1} N + K)$ time [11]. The second data structure, also by Chazelle, uses

*Work was supported in part by the Danish National Research Foundation and the Danish Strategic Research Council.

†Center for Massive Data Algorithmics, a center of the Danish National Research Foundation.

$O(N(\log N/\log\log N)^{d-1})$ space but has a slightly higher query time of $O(\log^{d-1+\varepsilon} N + K)$ [13] (in the same paper Chazelle proves that this space complexity is the best possible for any polylogarithmic query bound). The third data structure has the fastest query time, $O(\log^{d-2} N + K)$, but it occupies $O(N \log^d N)$ space [10], [17]. All the three high-dimensional data structures are essentially obtained using techniques that extend low-dimensional (restricted queries) structures to higher dimensions.

1.2. Restricted Queries

Let $Q(d, k)$, $0 \leq k \leq d$, denote the restricted case of the d-dimensional problem in which k of the dimensions have finite ranges. Refer to Fig. 1. The $Q(2, 1)$ and $Q(d, 0)$ problems are often *called 3-sided planar range reporting* and *dominance reporting*, respectively. The $Q(1, 0)$ problem can obviously be solved with optimal $O(\log N + K)$ query time and linear space. $Q(2, 1)$ can be solved in the same bounds using a priority search tree [24]. The $Q(3, k)$ problem can be solved with $O(N \log^k N)$ space and $O(\log N + K)$ query time [1], [10], [17].

Using a couple of general techniques, the above structures (along with structures for $Q(1, 1)$ and $Q(2, 2)$) can be used to obtain $Q(d, d)$ structures (including the ones discussed in Section 1.1). The first uses range trees and a data structure for $Q(d, k)$ to solve $Q(d + 1, k + 1)$, while incurring a $\log N$ factor increase in both space and query complexity [9]. We call this method *dimension reduction*. The second solves $Q(d, k + 1)$ by using a data structure for $Q(d, k)$ and paying a $\log N$ factor increase in the space complexity [17]. We call this *side reduction*. Note that this means that any improvements to lower-dimensional structures immediately carries over to higher dimensions.

1.3. Previous I/O-Model Results

External memory data structures are designed in an I/O-model where B elements are moved between main memory and disk in one I/O; computations can only occur on elements in the main memory of size M [4]. The goal is to answer a query using as few I/Os as possible.

In the I/O-model, $Q(1, 1)$ can be solved optimally in linear space and using $O(\log_B N + K/B)$ I/Os using a B-tree [18]. Similar to the pointer machine model, linear space structures with $O(\log_B N + K/B)$ query I/Os also exist for $Q(2, 1)$ [7]. For $Q(2, 2)$, the same query bound can be obtained with $O(N \log N/\log\log_B N)$ space, which is optimal [7]. Finally, the best data structure for $Q(3, k)$ uses $O(N \log^k N)$ space and answers queries in $O(\log_B N + K/B)$ I/Os [1]. This is optimal only for $k = 0$.

The dimension reduction and side reduction techniques are applicable in the I/O-model. However, they still incur a $\log_2 N$ (or rather $\log_2(N/B)$) factor increase in the query and/or space complexity, respectively. Thus higher-dimensional structures based on the two techniques are not explicitly mentioned in the external memory literature, since the $\log_2(N/B)$ factor seems far from optimal.

The best known lower bound for three- and higher-dimensional orthogonal range reporting in the I/O-model is due to Hellerstein et al. [23], who showed that $\Omega(N(\log B/\log\log_B N)^{d-1})$ space is needed to solve $Q(d, d)$. Note the $\log B$ (rather than $\log N$) denominator in this bound, and thus the large gap between the lower bound and the known $Q(3, 3)$ structure.

1.4. Our Results

Our main result is a reduction of the penalties suffered by side and dimension reductions to $\log N/\log\log N$ in the pointer machine model and to $\log N/\log\log_B N = \log_B N/\log_B\log_B N = \log_{\log_B N} N$ in the I/O-model, (with two-dimensional problems as base cases).

Our result has several immediate implications in the pointer machine model. The most significant one is a $Q(d, d)$ data structure with $O(N(\log N/\log\log N)^{d-1})$ space and $O(\log^{d-1} N/(\log\log N)^{d-2} + K)$ query time. This improves both data structures by Chazelle [11], [13] and achieves $o(\log^{d-1} N)$ query search time with optimal space. Alternatively, we can obtain a structure with $O(\log^{d-2} N + K)$ query time but using $O(N \log^d N/(\log\log N)^3)$ space, which improves the third known $Q(d, d)$ data structure. Our main result also gives the first optimal data structure for $Q(3, 1)$ with $O(\log N + K)$ query time using $O(N \log N/\log\log N)$ space.

In the I/O-model the impact of our main result is also significant. We show that $Q(d, d)$ can be solved with $O(N(\log N/\log\log_B N)^{d-1})$ space and $O(\log_B^{d-1} N/(\log_B\log_B N)^{d-2} + K/B)$ query I/Os. Another consequence is an optimal data structure for $Q(3, 1)$ with $O(\log_B N + K/B)$ query I/Os using $O(N \log N/\log\log_B N)$ space. In three dimensions, we also obtain a query optimal structures answering queries $Q(3, k)$ queries in $O(\log_B N + K/B)$ I/Os using $O(N(\log N/\log\log_B N)^k)$ space. For $k = 1$ the space bound is also optimal.

On the lower bound side, we show that any $Q(d, d)$

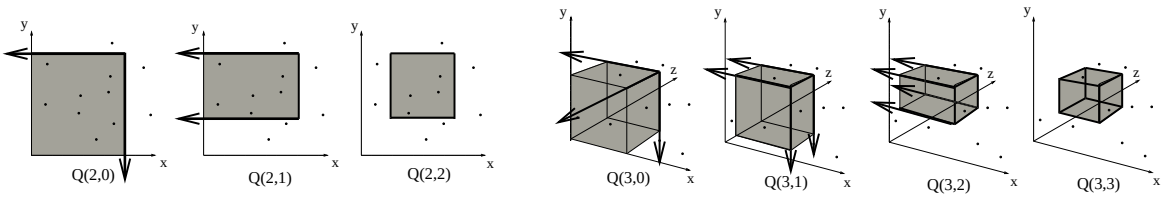

Figure 1. Two- and three-dimensional queries.

Queries	Space	Query bound	Ref./Notes	Deviation
Q(2,1)	N	$\log N + K$	PM, [24]	1 *(opt)*
Q(2,1)	N	$\log_B N + K/B$	IO, [7]	1 *(opt)*
Q(2,2)	$N \cdot \log_{\log N} N$	$\log N + K$	PM, [11]	1 *(opt)*
Q(2,2)	$N \cdot \log_{\log_B N} N$	$\log_B N + K/B$	IO, [7]	1 *(opt)*
Q(3,0)	N	$\log_B N + K/B$	IO, [1]	1 *(opt)*
Q(3,1)	$N \cdot \log N$	$\log N + K$	PM, [1], [17], [10]	$(\log \log N)$
Q(3,1)	$N \cdot \log N$	$\log_B N + K/B$	IO, [1]	$(\log \log_B N)$
Q(3,1)*	$\mathbf{N \cdot \log_{\log N} N}$	$\mathbf{\log N + K}$	**PM**	**1 *(opt)***
Q(3,1)*	$\mathbf{N \cdot \log_{\log_B N} N}$	$\mathbf{\log_B N + K/B}$	**IO**	**1 *(opt)***
Q(3,3)	$N \cdot \log^3 N$	$\log_B N + K/B$	IO, [1]	$(\log \log_B N)^3$
Q(3,3)*	$\mathbf{N \cdot \log^2_{\log N} N}$	$\mathbf{\log N \cdot \log_{\log N} N + K}$	**PM**	**1**
Q(3,3)*	$\mathbf{N \cdot \log^3_{\log N} N}$	$\mathbf{\log N + K}$	**PM**	**1**
Q(3,3)*	$\mathbf{N \cdot \log^2_{\log_B N} N}$	$\mathbf{\log_B N \cdot \log_{\log_B N} N + K/B}$	**IO**	**1**
Q(3,3)*	$\mathbf{N \cdot \log^3_{\log_B N} N}$	$\mathbf{\log_B N + K/B}$	**IO**	**1**
Q(d,d)	$N \cdot \log^{d-2} N \cdot \log_{\log N} N$	$\log^{d-1} N + K$	PM, [11]	$(\log \log N)^{2d-4}$
Q(d,d)	$N \cdot \log^{d-1}_{\log N} N$	$\log^{d-1+\varepsilon} N + K$	PM,[13]	$\log^{\varepsilon} N$
Q(d,d)	$N \cdot \log^d N$	$\log^{d-2} N + K$	PM, [1], [17], [10]	$(\log \log N)^{2d-3}$
Q(d,d)*	$\mathbf{N \cdot \log^3_{\log N} N \cdot \log^{d-3} N}$	$\mathbf{\log^{d-2} N + K}$	**PM**	$\mathbf{(loglog N)^{2d-6}}$
Q(d,d)*	$\mathbf{N \cdot \log^{d-1}_{\log N} N}$	$\mathbf{\log N \cdot \log^{d-2}_{\log N} N + K}$	**PM**	**1**
Q(d,d)*	$\mathbf{N \cdot \log^{d-1}_{\log_B N} N}$	$\mathbf{\log_B N \cdot \log^{d-2}_{\log_B N} N + K/B}$	**IO**	**1**

Table I. A summary of our upper bound results in bold as well as the best previous upper bounds on orthogonal range reporting. Our results are marked with an *. PM and IO stand for the pointer machine and I/O-model, respectively. Deviation measures the ratio of the space-query product, $S(N) \cdot Q(N)$, to $N \cdot \log_B^{d+k-2} N/(\log_B \log_B N)^{d+k-3}$ ($B = 2$ for the pointer machine model). Deviations marked with *(opt)* indicate that the result is *provably* optimal.

data structure with a query complexity polynomial in $\log_B N$ has to use $\Omega(N(\log N/\log\log_B N)^{d-1})$ space. This proves the space complexity of our $Q(d,d)$ data structure is optimal for a large range of values of B. A comparison of our results with the previous ones is given in Table I. Note that the Table displays a curious pattern: for $Q(d,k)$ the space $S(N)$ and the query search time $Q(N)$ of all the optimal results lie on the curve $S(N) \cdot Q(N) = N \log_B^{d+k-2} N/(\log_B \log_B N)^{d+k-3}$ ($B = 2$ for the pointer machine model). If this is the right trade-off curve for $S(N)$ and $Q(N)$, then all our main data structures are optimal!

In the next section we describe the dimension reduction technique and describe how it motivates a "concurrent" version of range searching where the same query needs to be answered on several different point sets. In Section 3, we show how to use concurrent range searching to obtain our improved versions of the dimension and side reduction techniques. Using these, we present our main orthogonal range searching results in Section 4. Finally, in Section 5 we prove our space lower bound. Conclusions and open problems are given in Section 6.

2. Dimension Reduction

Dimension reduction is the only known tool that allows us to solve orthogonal range searching in higher dimensions. As our ideas build upon this technique, we briefly describe it here. Consider the $Q(d+1, k+1)$ problem and assume we have access to a black-box solution $\mathcal{A}$ for $Q(d, k)$. Let $p_1, \ldots, p_N$ be the points of the input set S sorted increasingly according to the value of the last coordinate. Implement $\mathcal{A}$ on the projection of S onto the first d dimensions, then recurse on the sets $S_\ell := \{p_1, \ldots, p_{N/2}\}$ and $S_r := \{p_{N/2+1}, \ldots, p_N\}$. Let H be the hyperplane passing through $p_{N/2}$ and orthogonal to the vector $(0, 0, \ldots, 0, m_{N/2})$ where $m_{N/2}$ is the last coordinate of $p_{N/2}$. For the query q we have two cases: (i) If q completely lies at one side of H, then it is answered recursively using data structures implemented on S_ℓ or S_r. (ii) If q intersects H then it can be decomposed into two $Q(d+1, k)$ queries, one on each of the sets S_ℓ and S_r. Thus, it suffices to describe how to answer a $Q(d+1, k)$ query. Assume now that q is one such query. Again, we have two cases: (i) If q completely lies at one side of H, then it is answered recursively. (ii) If q intersects H then it can be decomposed into two queries; one query will be a $Q(d+1, k)$ query, which is answered recursively, but the other will be a $Q(d, k)$ query which can be answered directly by $\mathcal{A}$.

Let $S_{\mathcal{A}}(N)$ and $Q_{\mathcal{A}}(N)$ be the worst-case space and the query time of $\mathcal{A}$, respectively. The above solution takes $O(S_{\mathcal{A}}(N) \log N)$ space and has $O(Q_{\mathcal{A}}(N) \log N)$ query time. If we model this recursive construction with a tree, it becomes clear that the $\log N$ factor comes from the height of the tree. In other words, a tree of height h will result in an $O(hS_{\mathcal{A}}(N))$-space data structure with $O(hQ_{\mathcal{A}}(N))$ query time.

It is known that if a tree of fanout t is used instead of a binary tree (in other words, at each step the point set is partitioned into t sets rather than two), then the height of the tree will be $\log_t N$. For $t = \log^{\varepsilon} N$ this is equal to $\log N / \log\log N$. Unfortunately, answering queries using a tree of larger fanout is difficult as we might need to answer one query on up to t different point sets. Previously, three techniques were used to handle this issue. One incurs extra penalties in space, the other incurs extra penalties in query time and the last one only works in 2-d [11]. Thus, achieving factor $\log_t N$ penalty in both space and query was left open. This barrier is also responsible for the lack of I/O efficient results in higher dimensions.

3. Concurrent $Q(d, k)$ Problem

The motivation for concurrent range reporting is clear in the light of the previous discussion: we want to solve all the t queries generated by a tree of fanout t at the same time. The caveat is that the obvious definition fails to provide us with any means of attack.

We formally define the concurrent range searching problem as follows. Let S be an input set of N points and assume each point $p \in S$ has been assigned a color $\mathcal{A}(p)$ from a set C of colors. A *concurrent* $Q(d, k)$ problem is defined by a set $\mathcal{P} \subset 2^C$ (intuitively, the set of all the "possible" sets of colors) and is an orthogonal range reporting problem where the query is a tuple (q, L) in which $L \in \mathcal{P}$ and q is a $Q(d, k)$ query. The output should be the set of all the points p such that $\mathcal{A}(p) \in L$ and $p \in q$. Note that this should not be confused with the usual colored range searching where we are interested in the set of *colors* in q and not the points (e.g., see [22]). We use $Q_{\mathcal{A},C,\mathcal{P}}(d, k)$ to denote this concurrent $Q(d, k)$ problem.

Our main results of this section are the dimension and side reduction techniques for concurrent orthogonal range reporting and a solution for concurrent $Q(2, 1)$.

3.1. Side Reduction

Consider the $Q_{\mathcal{A},C,\mathcal{P}}(d+1, k+1)$ problem. We describe our reduction only for queries that have two sides at the last dimension; otherwise, we can apply basic geometric transformations on the input set and use the solution for this specific case. This will increase the space by a constant factor only.

Build a balanced tree T of fanout t on the point set sorted by their last coordinates. Let v be a node in T and $c_1, \ldots, c_t$ be the children of v. Now, v corresponds to an interval $[a_v; b_v)$ on the last dimension and the last coordinate of all the points in the subtree rooted at v, $T(v)$, lie between a_v and b_v; the children of v further subdivide this interval into t disjoint smaller intervals. We define C_v as $C \times \{1, \ldots, t\}$ and $\mathcal{P}_v$ as the set containing all the sets $L_{L,i,j} := \{(c, k) | c \in L, i \le k \le j\}$ for all $L \in \mathcal{P}$ and all $0 \le i \le j \le t$. Clearly, $|C_v| = t|C|$ and $|\mathcal{P}_v| \le t^2|\mathcal{P}|$. For the points $p \in T(v)$ we define a new color assignment $\mathcal{A}_v$ by setting $\mathcal{A}_v(p) = (\mathcal{A}(p), j)$ where j is the index of the subset $T(c_j)$ that contains p. We implement a data structure $\mathcal{D}_v$ for $Q_{\mathcal{A}_v,C_v,\mathcal{P}_v}(d+1, k)$ on $T(v)$. We repeat this operation for every node in T.

To answer the query (q, L), let $[a; b]$ be the projection of q on the last dimension. We start from the root and walk down the tree to find the first node v such that $[a; b]$ is not contained in the interval of one child of v. Let $c_1, \ldots, c_t$ be the children of v and

assume $[a;b]$ intersects the intervals of $c_i, \ldots, c_j$, for $1 \le i < j \le t$. Finding v costs $Q_{\text{search}}(N)$. We have $a_{c_i} \le a < b_{c_i} \le a_{c_j} \le b < b_{c_j}$. We create two new boxes, q_r and q_ℓ by modifying the coordinates of q: q_r is made by setting a to $-\infty$ and q_ℓ is made by setting b to $+\infty$. Next, we define two lists of colors, $L_\ell := \{(c,x) | c \in L, i \le x \le j-1\}$ for q_ℓ and $L_r := \{(c,j) | c \in L\}$ for q_r. The pairs (q_r, L_r) and (q_ℓ, L_ℓ) form two valid $Q(d+1,k)$ concurrent queries since both L_ℓ and L_r are in $\mathcal{P}_v$. We query both on $\mathcal{D}_v$.

Correctness: We must report all the points p such that $p \in q$ and $\mathcal{A}(p) \in L$. Consider a point p and let z be the value of its last coordinate. We have four possible cases:

1) $\mathcal{A}(p) \notin L$: In this case $\mathcal{A}_v(p) \notin L_r$ and $\mathcal{A}_v(p) \notin L_\ell$ and thus p will not be reported. From now on, we assume $\mathcal{A}(p) \in L$.
2) $p \in T(c_j)$: In this case $\mathcal{A}_v(p) = (\mathcal{A}(p), j)$, so $\mathcal{A}_v(p) \in L_r$ and $\mathcal{A}_v(p) \notin L_\ell$. Clearly, the query (q_ℓ, L_ℓ) will not report p but (q_r, L_r) will report p if and only if p is contained in q_r. However, if p is in q_r then $z \le b$. Note that we know $a_{c_j} \le z$ since $p \in T(c_j)$ and thus $a < z$. These imply p will be reported if and only if it is contained in q.
3) $p \in \cup_{i \le x \le j-1} T(c_x)$: Let x be the index such that $T(c_x)$ contains p. As with the previous case, we have $\mathcal{A}_v(p) = (\mathcal{A}(p), x) \in L_\ell$ and $\mathcal{A}_v(p) \notin L_r$. Similarly, since $z < b_{c_{j-1}} \le b$, p will be reported if and only if it is contained in q.
4) $p \notin \cup_{i \le x \le j} T(c_x)$: This is the else case to the above three. It is clear that in this case $p \notin q$ and since $\mathcal{A}_v(p) \notin L_r \cup L_\ell$ none of the queries will report p.

Thus, we have the following result.

Lemma 1. *$Q_{\mathcal{A},C,\mathcal{P}}(d+1,k+1)$ can be solved with $O\left(N + hS_{n_c,n_p,d+1,k}(N)\right)$ space and with the query complexity of $O\left(Q_{\text{search}}(N) + Q_{n_c,n_p,d+1,k}(N)\right)$. Here, h is the height of T, $S_{n_c,n_p,d+1,k}(\cdot)$ and $Q_{n_c,n_p,d+1,k}(\cdot)$ are the worst-case query and space complexity of the concurrent $Q(d+1,k)$ data structure used, respectively, $n_c = \max_{v \in T} |C_v|$, $n_p = \max_{v \in T} |\mathcal{P}_v|$, and $Q_{\text{search}}(\cdot)$ is the cost of search for a node in the tree.*

3.2. Dimension Reduction

We describe the dimension reduction for concurrent $Q(d+1,k+1)$ queries. We assume the query has two sides at the last dimension.

Build a balanced tree T of fanout t on the last dimension and consider the notations introduced in the previous subsection and build $C_v, \mathcal{P}_v$ and $\mathcal{A}_v$ similarly. The difference is that this time we project the points of $T(v)$ onto the first d dimensions and implement a data structure $\mathcal{D}_v$ for $Q_{\mathcal{A}_v,C_v,\mathcal{P}_v}(d,k)$ on the projected points.

To answer the query (q,L), let $[a;b]$ be the projection of q on the last dimension. Find the first node v_1 such that $[a;b]$ is not contained in the interval of one child of v_1. Assume $[a;b]$ intersects the intervals of $c_i, \ldots, c_j$, for $1 \le i < j \le t$. This time, we create three boxes q_ℓ, q_m and q_r by modifying coordinates of q. q_ℓ is obtained by setting b to $+\infty$, q_r is obtained by setting a to $-\infty$, and q_m is a d-dimensional box obtained by setting $a = -\infty$ and $b = +\infty$. We define the color list L_m as $\{(c,x) | c \in L, i < x < j\}$ and query $\mathcal{D}_v$ with (q_m, L_m). Boxes q_ℓ and q_r define two v_1 to leaf query paths (to be described); we only analyze the query path for q_r below, as the other one is similar.

Define $v_2 := c_j$. Now (q_r, L) is a concurrent $Q(d+1,k)$ query on v_2 and we answer it as follows. Let $c_1^{(2)}, \ldots, c_t^{(2)}$ be the children of v_2 and assume x is the index such that $a_{c_x^{(2)}} \le b < b_{c_x^{(2)}}$. We create a color list $L_m^{(2)} := \{(c,y) | c \in L, 1 \le y \le x-1\}$ and query $(q_m, L_m^{(2)})$ on D_{v_2}. Now we set $v_3 := c_x^{(2)}$ and continue until we reach a leaf.

Correctness: An analysis similar to that of side reduction shows (q_m, L_m) and $(q_m, L_m^{(2)})$ respectively return all the points in $q \cap (T(c_{i+1}) \cup \cdots \cup T(c_{j-1}))$ and $q \cap (T(c_1^{(2)}) \cup \cdots \cup T(c_{x-1}^{(2)}))$ with the right colors and nothing more. The recursive call to v_3 takes care of the points in $T(c_x^{(2)})$. Finally, the same analysis holds for the other query path determined by q_ℓ. Thus, we have the following lemma.

Lemma 2. *$Q_{\mathcal{A},C,\mathcal{P}}(d+1,k+1)$ can be solved with $O\left(N + hS_{n_c,n_p,d,k}(N)\right)$ space and with the query complexity of $O\left(Q_{\text{search}}(N) + hQ_{n_c,n_p,d,k}(N)\right)$. Here, h is the height of T, $S_{n_c,n_p,d,k}(\cdot)$ and $Q_{n_c,n_p,d,k}(\cdot)$ are the worst-case query and space complexity of the concurrent $Q(d,k)$ data structure used, respectively, $n_c = \max_{v \in T} |C_v|$, $n_p = \max_{v \in T} |\mathcal{P}_v|$, and $Q_{\text{search}}(\cdot)$ is the cost of search for a node in the tree.*

3.3. Answering Concurrent $Q(2,1)$ Queries

Here we solve the concurrent $Q_{\mathcal{A},C,\mathcal{P}}(2,1)$ problem. As this will be used at the base case for our higher dimensional results, any suboptimal space or query bound will carry over to higher dimensions. Thus, it is crucially important that we obtain an efficient structures. Our main result of this subsection is the following.

Lemma 3. *$Q_{\mathcal{A},C,\mathcal{P}}(2,1)$ can be solved with $O(N)$ space and $O(|\mathcal{P}||C| + \log N + K)$ query time in a*

pointer machine and $O(|\mathcal{P}||C|+\log_B N+K/B)$ query I/Os in the I/O-model.

Proof: We only provide a data structure for the I/O-model. By setting $B = 2$ it can be turned into a data structure for a pointer machine. We also assume that the three-sided query is in the form of $[x_1; x_2] \times (-\infty, y]$; general three-sided queries can be reduced to this case.

Sort the points according to their x-coordinates and partition the point set into $\frac{N}{\alpha}$ vertical slabs, where $\alpha := B|C||\mathcal{P}|$, such that each slab contains α points. For each slab b we build a basic data structure $D_{\text{basic}}(b)$ as follows: for every color c in b, the points with color c are stored increasingly according to their y-coordinates in consecutive blocks (linked list in a pointer machine).

Next, for every set $L \in \mathcal{P}$, we build a search structure of *sublinear* size which we denote by $D_{\text{search}}(L)$. To do this, first we copy the basic data structure but we delete some of its points in two stages. First, we delete all points p such that $\mathcal{A}(p) \notin L$. Let $p_{c,1}, p_{c,2}, \ldots, p_{c,t_c}$ be the remaining points in slab b, sorted increasingly according to their y-coordinates. Next we delete all the points $p_{c,i}$, $i > B$. The $D_{\text{search}}(L)$ is made by implementing an optimal I/O efficient $Q(2, 1)$ structure on the remaining points. Finally, we place a pointer from $p_{c,B}$ (if it exists) to its copy in $D_{\text{basic}}(b)$.

Each slab b contains α points but at most $B|L| \leq B|C|$ points are stored in each slab in $D_{\text{search}}(L)$. Thus, the size of $D_{\text{search}}(L)$ is $O\left(\frac{N}{\alpha}B|C|\right) = O(N/|\mathcal{P}|)$. We build $|\mathcal{P}|$ different search structures (one for each list in $\mathcal{P}$), so our space complexity is linear.

Let (q, L) be the query and $b_i, \ldots, b_j$ be the slabs intersected by q. We decompose q into three smaller boxes, q_ℓ, q_m and q_r such that q_ℓ and q_r are contained in b_i and b_j respectively and $q_m := q \setminus (q_\ell \cup q_r)$. We answer (q_ℓ, L) and (q_r, L) by scanning all the points in b_i and b_j and this takes $O(\alpha/B) = O(|C||\mathcal{P}|)$ I/Os. To answer (q_m, L), using $D_{\text{search}}(L)$ first we find all the points of $D_{\text{search}}(L)$ that are inside q_m. This costs $O(\log_B N + K'/B)$ I/Os where K' is the number of points reported. A helpful observation is that the set of points reported is a subset of the final output. During this operation, whenever we encounter a point $p_{c,B}$ that is to be reported, we follow the pointer to D_{basic} and scan all the points stored in the following blocks until the y-coordinates of the stored points exceed that of q_m. The crucial observation is that the cost of the pointer jump can be charged to the output size (to the B points of color c that have been outputted from the slab containing $p_{c,B}$). It is easy to check that all the relevant points are reported and that the cost of the query is $O(|C||\mathcal{P}| + \log_B N + K/B)$ I/Os. ■

4. Answering $Q(d, d)$ Queries

The $Q(d, d)$ problem is a concurrent $Q(d, d)$ problem with $C = \{1\}$, $\mathcal{P} = \{\{1\}\}$ and $\mathcal{A}(p) = 1$. To solve it, we apply our reductions outlined in Lemmas 1 and 2.

After $d - 2$ applications of dimension reduction, $Q(d, d)$ will be reduced to a concurrent $Q(2, 2)$ problem. Notice that each dimension reduction increases the size of the sets C and $\mathcal{P}$ by factors of t and t^2 respectively. Next we apply a side reduction which further increases C and $\mathcal{P}$ by factors of t and t^2. Thus, at the end, our subproblems will consist of $Q_{\mathcal{A},C,\mathcal{P}}(2, 1)$ problems where $|C| = t^{d-1}$ and $|\mathcal{P}| = t^{2d-2}$.

We set $t := \log_B^{1/(3d-3)} N$. This means in all our subproblems we have $|C||\mathcal{P}| = O(\log_B N)$. By Lemma 3 these can be solved with linear space and $O(\log_B N + K/B)$ I/Os. As the height of the trees used in our constructs is $\log_t N$, we have $h = \log N/\log\log_B N$. By Lemma 1, our side reduction adds a factor of h to space. By Lemma 2, our $d - 2$ dimension reductions add a factor of h^{d-2} to both space and query. Thus we obtain the following results.

Theorem 1. *In the I/O-model, the orthogonal range reporting problem on N input points can be solved with $O(N(\log N/\log\log_B N)^{d-1})$ space and $O(\log_B N(\log N/\log\log_B N)^{d-2}+K/B)$ query I/Os.*

Theorem 2. *In a pointer machine, the orthogonal range reporting problem on N input points can be solved with $O(N(\log N/\log\log N)^{d-1})$ space and $O(\log^{d-1} N/(\log\log N)^{d-2} + K)$ query time.*

The space usage of Theorem 2 is optimal by [13]. Using our techniques we can also obtain the first optimal result for $Q(3, 1)$ queries.

Theorem 3. *$Q(3, 1)$ queries can be solved optimally in the pointer machine model (resp. the I/O-model) using $O(N \log N/\log\log N)$ space (resp. $O(N \log N/\log\log_B N)$ space) and with $O(\log N + K)$ query time (resp. $O(\log_B N + K/B)$ query I/Os).*

Proof: Use the traditional side reduction technique with a tree of fanout $t := \log^{1/3} N$. As discussed, this results in two $Q(3, 0)$ queries and up to t $Q(2, 0)$ queries. The $Q(3, 0)$ queries can be solved by the linear-space data structures outlined in [1]. The $Q(2, 0)$ queries can be solved using Lemma 3. The space bound is optimal since with a simple geometric transformation one can use a data structure for $Q(3, 1)$ to answer $Q(2, 1)$ queries and an $\Omega(N \log N/\log\log N)$ (resp. $\Omega(N \log N/\log\log_B N)$) space lower bound is known for $Q(2, 1)$ [7], [13]. ■

Theorem 4. *For $k = 2, 3$, $Q(3, k)$ can be*

solved in pointer machine model (resp. I/O-model) using $O(N(\log N/\log\log N)^k)$ space (resp. $O(N(\log N/\log\log_B N)^k)$ space) and with optimal queries.

Proof: The proof is similar to that of Theorem 3: by normal side reduction, we get two $Q(3, k-1)$ queries and one concurrent $Q(2, k-1)$ query which can be solved by their corresponding theorems outlined above. ■

Corollary 1. *Orthogonal range reporting can be solved with $O(N\log^d N/(\log\log N)^3)$ space and $O(\log^{d-2} N + K)$ query time.*

5. I/O-model Lower Bound

In this section we use the indexability theory of Hellerstein et al. [23] to prove that any data structure answering $Q(d, d)$ queries in the I/O-model using $O(\log_B^c N + K/B)$ query I/Os for any constant $c > 0$, has to use $\Omega(N(\log N/\log\log_B N)^{d-1})$ space.

In the indexability model [23] an indexing problem is described by a *workload* $W = (I, Q)$, where I is a set of input elements and Q is a set of subsets of I; the elements of Q are called *queries*. Given a workload W and a block size B, an *indexing scheme* S is defined on I by a block assignment function, $\mathbb{B}$, which is a set of B-sized subsets of I. Intuitively, all the elements in a set $b \in \mathbb{B}$ are stored in one block.

The quality of an indexing scheme is quantified by two parameters: *redundancy* and *access overhead*. The redundancy r of S is a measure of the space overhead and is defined as $r = B|\mathbb{B}|/|I|$. If any query in Q is covered by at most $A_0 + A_1\lceil |q|/B\rceil$ blocks of $\mathbb{B}$, then the access overhead is defined as the (A_0, A_1) tuple [8] (this a slight variation on the original definition of access overhead [23]). For any data structure in the I/O-model, an indexing scheme is naturaly defined by just looking at the points stored in the blocks of the storage medium. The following *redundancy theorem* relates redundancy and access overhead and is the main tool for proving space lower bounds in the I/O-model [8], [23].

Theorem 5 (Refined Redundancy Theorem [8])**.** *For a workload $W = (I, Q)$ with $|I| = N$ and where $Q = \{q_1, q_2, \ldots, q_m\}$, let $S = (I, \mathbb{B})$ be an indexing scheme for W with access overhead (A_0, A_1) with $A_1 \le \sqrt{B}/8$ such that for any $1 \le i, j \le m, i \ne j : |q_i| \ge BA_0$ and $|q_i \cap q_j| \le B/(64A_1^2)$. Then the redundancy of S is bounded by $r \ge \frac{1}{12N}\sum_{i=1}^m |q_i|$.*

Consider a data structure for $Q(d, d)$ with $c_0\log_B^c N + c_1\frac{T}{B}$ query bound where c_0 and c_1 are constants. We choose $A_0 = c_0\log_B^c N$ and $A_1 = c_1$. The refined redundancy theorem then states that if we can construct a set of N points and m query boxes $q_1, \ldots, q_m$, such that any box contains at least BA_0 points and where the intersection of any pair of boxes contains at most $B/(64c_1^2)$ points, then the amount of space needed by any data structure is $\Omega(\sum_{i=1}^m |q_i|)$. The goal is thus to maximize the sum of the sizes of the queries.

In two-dimensions, the $\Omega(N\log N/\log\log_B N)$ space lower bound for $Q(2, 2)$ was obtained using a *Fibonacci workload*. However, generalizing the Fibonacci workload to higher dimensions seems hard, and the previously best known d-dimensional $\Omega(N(\log B/\log\log_B N)^{d-1})$ space lower bound instead utilizes a simple pointset consisting of a $N^{1/d} \times \cdots \times N^{1/d}$ grid. In internal memory, the space lower bounds in the pointer machine model [28] for d-dimensional range searching were proven by Chazelle [13]. In 2-d, a fairly simple point-set (workload) was used to prove the bounds, whereas a much more complex point-set and a randomized argument were used in higher dimensions.

Here we generalize Chazelle's planar point set to higher dimensions using a deterministic construction. Such a deterministic generalization was given by Chazelle as well, but for the off-line orthogonal range searching in the semi-group model [15]. In fact, by modifying the parameters used in his proof, one can prove the $\Omega(N(\log N/\log\log_B N)^{d-1})$ space lower bound; however, the lower bound will be valid only if $B = O(\log_B N)$, which is an unrealistic assumption in the I/O-model. Relaxing this constraint seems to require more substantial changes, e.g., changing the query or the point set. Here we present an alternate but similar construction that achieves this. Our lower bound holds for $2 \le B \le \sqrt{M} \le \sqrt{N}$, known also as the *tall-cache assumption*, which is a much more reasonable assumption.

Point set I.: Let $a_1 = 64A_0A_1^2$ and $a_j = (\prod_{i=1}^{j-1} a_i) + 1$ for $j = 2, \ldots, d-1$. It is easily verified that $a_1, a_2, \ldots, a_{d-1}$ are relatively prime. We define the point set $I := \{(p_{a_1}(i), p_{a_2}(i), \ldots, p_{a_{d-1}}(i), i) \mid i = 0, 1, \ldots, N-1\}$, where $p_{a_j}(i)$ is obtained by first writing i in base a_j, then removing all but the $\lfloor \log_{a_j} N^{\frac{1}{4d}} \rfloor$ least significant digits (adding leading 0-digits if necessary), and finally reversing all the digits of the constructed number. We will use $\overleftarrow{m_{k-1}\ldots m_0}$ to denote the reversal of $m_{k-1}\ldots m_0$, that is, $\overleftarrow{m_{k-1}\ldots m_0} = m_0\ldots m_{k-1}$. The following lemma is an easy consequence of the defitions given above.

Lemma 4. *Consider the i'th point $p_i = (p_{a_1}(i), \ldots, p_{a_{d-1}}(i), i)$ in I. The k most significant*

digits of the j'th coordinate $p_{a_j}(i)$ are precisely $\overleftarrow{i \mod a_j^k}$ for $k \leq \lfloor \log_{a_j} N^{\frac{1}{4d}} \rfloor$.

Let X be the box in the positive quadrant anchored at the origin $(0,0,\ldots,0)$ with dimensions $a_1^{\lfloor \log_{a_1} N^{1/4d} \rfloor} \times \cdots \times a_{d-1}^{\lfloor \log_{a_{d-1}} N^{1/4d} \rfloor} \times N$; X contains all points in I. Now consider a box q inside X, and let $[x_1; x_2]$ be the range it spans in the j'th dimension. If $x_1 = m_0 \ldots m_{k-1} 00 \ldots 0$ and $x_2 = m_0 \ldots m_{k-1}(a_j - 1)(a_j - 1) \ldots (a_j - 1)$ in base a_j for some $m_0 \ldots m_{k-1}$, it follows from Lemma 4 that each point p_i with $i \mod a_j^k = \overleftarrow{m_0 \ldots m_{k-1}}$ has the j'th coordinate in the range $[x_1; x_2]$. If the same holds for each of the first $d-1$ dimensions, we can determine whether a point is inside q simply by looking at its d'th coordinate.

Query set Q.: Consider the set R consisting of one box with each of the following dimensions $a_1^{i_1} \times a_2^{i_2} \times \cdots \times a_{d-1}^{i_{d-1}} \times BA_0 a_1^{k_1} a_2^{k_2} \ldots a_{d-1}^{k_{d-1}}$ for $i_j \in \{0, \ldots, \lfloor \log_{a_j} N^{\frac{1}{4d}} \rfloor\}$ and $k_j = \lfloor \log_{a_j} N^{\frac{1}{4d}} \rfloor - i_j$.

Lemma 5. *Any $r \in R$ placed at the origin in d-dimensional space is completely contained in X. Furthermore, $|R| = \Omega((\log N / \log A_0 A_1^2)^{d-1})$*

Proof: The first $d-1$ dimensions of r are obviously within X. Using the tall-cache assumption we get that $BA_0 = Bc_0 \log_B^c N \leq \sqrt{N} c_0 \log_B^c N \leq \sqrt{N} c_0 \log_2^c N$ which for N greater than some constant is at most $\sqrt{N} N^{\frac{1}{4}} = N^{\frac{3}{4}}$. Thus the size of the d'th dimension of r is bounded by $BA_0 \cdot \prod_{i=1}^{d-1} a_i^{\lfloor \log_{a_i} N^{1/4d} \rfloor} \leq N^{\frac{3}{4}} \cdot (N^{\frac{1}{4d}})^{d-1} \leq N$, and therefore r fits within X.

To see the bound on the size of R, simply count the number of combinations of i_j in the definition of R

$$|R| = \prod_{i=1}^{d-1} \lfloor \log_{a_i} N^{\frac{1}{4d}} \rfloor + 1 \geq \prod_{i=1}^{d-1} \log_{a_i} N^{\frac{1}{4d}} \geq \left(\log_{a_{d-1}} N^{\frac{1}{4d}} \right)^{d-1} \geq \left(\log_{a_1^{2^d}} N^{\frac{1}{4d}} \right)^{d-1} = \left(\log_{a_1} N^{\frac{1}{4d \cdot 2^d}} \right)^{d-1} = \left(\frac{\log_{a_1} N}{4d \cdot 2^d} \right)^{d-1} = \frac{\left(\log_{a_1} N\right)^{d-1}}{(4d)^{d-1} \cdot 2^{d^2-d}} = \Omega\left(\left(\log_{a_1} N \right)^{d-1} \right) = \Omega\left(\left(\log_{A_0 A_1^2} N \right)^{d-1} \right) = \Omega\left(\left(\frac{\log N}{\log A_0 A_1^2} \right)^{d-1} \right)$$

■

Our query set Q consists of the boxes obtained by tiling X with each of the boxes $r \in R$ in turn, starting at the origin. Notice that we will use only those queries that are completely contained in X. Refer to Fig. 2.

Lemma 6. *For any query $q \in Q$, $|q| = BA_0$.*

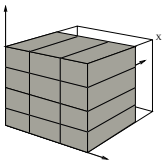

Figure 2. Tiling X with a box $r \in R$. Note that r might not tile X completely in all dimensions.

Proof: Let q be a box in Q with dimensions $a_1^{i_1} \times \cdots \times a_{d-1}^{i_{d-1}} \times BA_0 a_1^{k_1} \ldots a_{d-1}^{k_{d-1}}$, and consider its j'th dimension ($j < d$). Since q was placed by tiling from the origin, q will span the range $I_j = [c_j a_j^{i_j}; (c_j+1) a_j^{i_j})$ in the j'th dimension for some $c_j = m_0 m_1 \ldots m_{k_j - 1}$, where c_j is written in base a_j. From Lemma 4 it then follows that the i'th point $p_i = (p_{a_1}(i), \ldots, p_{d-1}(i), i)$ of I is inside q if and only if

$$\forall\, 1 \leq j \leq d-1, \qquad \mod a_j^{k_j} = \overleftarrow{c_j} \tag{1}$$

and

$$c_d BA_0 a_1^{k_1} \ldots a_{d-1}^{k_{d-1}} \leq i < (c_d + 1) BA_0 a_1^{k_1} \ldots a_{d-1}^{k_{d-1}}.$$

As $a_1, \ldots, a_{d-1}$ are relatively prime and $\gcd(a, b) = 1$, by the Chinese Remainder Theorem there is a unique value of i modulo $a_1^{k_1} a_2^{k_2} \ldots a_{d-1}^{k_{d-1}}$ that satisfies all of the $d-1$ requirements in (1). Since $q \subset X$, it follows from the last requirement on i that q contains precisely $(BA_0 a_1^{k_1} \ldots a_{d-1}^{k_{d-1}}) / (a_1^{k_1} \ldots a_{d-1}^{k_{d-1}}) = BA_0$ points. ■

Having defined our workload $W = (I, Q)$, we now bound the number of points in the intersection of any two query boxes in Q.

Lemma 7. *For any two query boxes $q_1, q_2 \in Q$, $|q_1 \cap q_2| \leq B/(64A_1^2)$*[1].

Proof: If q_1 and q_2 have the same dimensions, we get from the tiling that $q_1 \cap q_2 = \emptyset$ and the lemma follows. Now consider the case where q_1 and q_2 differ in at least one dimension. Let $a_1^{i_1} \times \cdots \times a_{d-1}^{i_{d-1}} \times BA_0 a_1^{k_{(i,1)}} \ldots a_{d-1}^{k_{(i,d-1)}}$ be the dimensions of q_1 and $a_1^{j_1} \times \cdots \times a_{d-1}^{j_{d-1}} \times BA_0 a_1^{k_{(j,1)}} \ldots a_{d-1}^{k_{(j,d-1)}}$ be the dimensions of q_2. Let $l < d$ be any dimension where $i_l \neq j_l$. W.l.o.g we assume that $i_l > j_l$. Since $a_l^{i_l}$ is just a multiplicative of $a_l^{j_l}$, it follow from the tiling that the intersection of q_1 and q_2 is either empty in the l'th dimension, or spans the same range as q_2. If the range is empty, our proof is done, so assume it equals the range of q_2. Now consider the box J that spans exactly the same ranges as q_1, except in the l'th dimension, where

[1]This property is one of the main differences between our point set and the one developed by Chazelle; his construction ensures that $|q_1 \cap q_2| = O(1)$ which is a more strict condition than ours and thus his bound is valid for small values of B.

it spans the same range as q_2. Clearly $q_1 \cap q_2 \subset J$. Using the Chinese Remainder Theorem, we get that J contains at most

$$\frac{BA_0 a_1^{k_{(i,1)}} \dots a_{d-1}^{k_{(i,d-1)}}}{a_1^{k_{(i,1)}} \dots a_l^{k_{(j,l)}} \dots a_{d-1}^{k_{(i,d-1)}}} \leq \frac{BA_0}{a_l} \leq \frac{B}{64A_1^2}$$

points. Since $q_1 \cap q_2 \subset J$, we have that $|q_1 \cap q_2| \leq |J| \leq B/(64A_1^2)$ and the lemma follows. ■

We now apply the redundancy theorem. By Lemma 6 and Lemma 7, our workload $W = (I, Q)$ fulfills the requirements of the Refined Redundancy Theorem. Thus, the redundancy, r, of any solution for this workload is at least $\frac{1}{12N}\sum |q_i|$.

Now consider any box $s \in R$. By Lemma 5 we know that s will be contained in X if placed at the origin. We also get from the definition of R, that the $d-1$ first dimensions of X are multiplicative of the $d-1$ first dimensions of s. It then follows from the tiling that every point in I will have its $d-1$ first coordinates inside one query box for every $s \in R$, and at least half the points will have their d'th coordinate inside one query box for every $s \in R$. Therefore $\sum |q_i| \geq |R|\frac{N}{2}$. Plugging this value in the bound for redundancy and using Lemma 5 implies $r = \Omega(|R|) = \Omega((\log N/\log A_0 A_1^2)^{d-1})$. Since $A_0 = c_0 \log_B^c N$ and $A_1 = c_1$, we obtain the desired lower bound.

Theorem 6. *There exist a workload (i.e., a set of points and a set of queries) W for $Q(d,d)$ such that any data structure for W that can answer queries in $O(\log_B^c N + \frac{K}{B})$ I/Os for any constant $c > 0$, requires $\Omega(N(\log N/\log\log_B N)^{d-1})$ space.*

6. Conclusion

In this paper we improved the dimension and side reduction techniques and thus obtained new orthogonal range reporting data structures in both the pointer machine model and the I/O-model. In the latter model we also provided a space lower bound. Our reductions incur $\log N/\log\log N$ (resp. $\log N/\log\log_B N$) factor penalties in space and/or query complexity in the pointer machine model (resp. the I/O-model).

Note that in the I/O-model, the penalties are minimized for $B = 2$ and for large values of B are far away from $\log_B N$. Still, for all our structures the overall query complexity decreases with B.

We believe it is unlikely that the techniques (specially the dimension reduction) can be further improved. Focussing on the I/O-model for example, our space lower bound proves that it is impossible to achieve $o(\log N/\log\log_B N)$ space penalty in dimension reduction even with $O(\log^{O(1)} N)$ penalty in query. Currently there are no query lower bounds available but as the source of both query and space penalties are the same (namely, the height of the tree used in the reduction), we suspect lowering the query penalty (without increasing the space penalty) needs completely new ideas.

Finally, we believe this work brings up many interesting open problems, including the following:

Open Problem. *What is the right trade-off curve for the query time and the space bound of the best possible $Q(d,k)$ data structures in the pointer machine model or the I/O-model?*

References

[1] P. Afshani, "On dominance reporting in 3D," in *ESA'08: Proc. of the 16th conference on Annual European Symposium*, 2008, pp. 41–51.

[2] P. K. Agarwal, "Range searching," in *CRC Handbook of Discrete and Computational Geometry*, J. E. Goodman and J. O'Rourke, Eds. CRC Press, Inc., 2004.

[3] P. K. Agarwal and J. Erickson, "Geometric range searching and its relatives," in *Advances in Discrete and Computational Geometry*, B. Chazelle, J. E. Goodman, and R. Pollack, Eds. AMS Press, 1999.

[4] A. Aggarwal and J. S. Vitter, "The input/output complexity of sorting and related problems," *Commun. ACM*, vol. 31, pp. 1116–1127, 1988.

[5] S. Alstrup, G. S. Brodal, and T. Rauhe, "New data structures for orthogonal range searching," in *FOCCS'00: Proc. of the 41st Annual Symposium on Foundations of Computer Science*, 2000, pp. 198–207.

[6] L. Arge, "External memory data structures," in *Handbook of Massive Data Sets*, J. Abello, P. M. Pardalos, and M. G. C. Resende, Eds. Kluwer Academic Publishers, 2002, pp. 313–358.

[7] L. Arge, V. Samoladas, and J. S. Vitter, "On two-dimensional indexability and optimal range search indexing," in *PODS '99: Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. New York, NY, USA: ACM, 1999, pp. 346–357.

[8] L. Arge, V. Samoladas, and K. Yi, "Optimal external-memory planar point enclosure," in *ESA'04: Proc. of the 12th conference on Annual European Symposium*, 2004, pp. 40–52.

[9] J. L. Bentley, "Multidimensional divide-and-conquer," *Communications of the ACM*, vol. 23, no. 4, pp. 214–229, 1980.

[10] P. Bozanis, N. Kitsios, C. Makris, and A. Tsakalidis, "New results on intersection query problems," *The Computer Journal*, vol. 40, pp. 22–29, 1997.

[11] B. Chazelle, "Filtering search: a new approach to query answering," *SIAM Journal on Computing*, vol. 15, no. 3, pp. 703–724, 1986.

[12] ——, "Functional approach to data structures and its use in multidimensional searching," *SIAM Journal on Computing*, vol. 17, no. 3, pp. 427–462, 1988.

[13] ——, "Lower bounds for orthogonal range searching: I.

the reporting case," *Journal of the ACM*, vol. 37, no. 2, pp. 200–212, 1990.

[14] ——, "Lower bounds for orthogonal range searching: part II. the arithmetic model," *Journal of the ACM*, vol. 37, no. 3, pp. 439–463, 1990.

[15] ——, "Lower bounds for off-line range searching," in *STOC '95: Proc. of the 27th annual ACM symposium on Theory of computing*, 1995, pp. 733–740.

[16] B. Chazelle and L. J. Guibas, "Fractional cascading: I. A data structuring technique," *Algorithmica*, vol. 1, pp. 133–162, 1986.

[17] ——, "Fractional cascading: II. Applications," *Algorithmica*, vol. 1, pp. 163–191, 1986.

[18] D. Comer, "The ubiquitous B-tree," *ACM Computing Surveys*, vol. 11, no. 2, pp. 121–137, 1979.

[19] M. L. Fredman, "The inherent complexity of dynamic data structures which accommodate range queries," in *FOCS '80: Proc. of the 21st Annual Symposium on Foundations of Computer Science*, October 1980, pp. 191–199.

[20] H. N. Gabow, J. L. Bentley, and R. E. Tarjan, "Scaling and related techniques for geometry problems," in *STOC '84: Proc. of the 16th annual ACM symposium on Theory of computing*, 1984, pp. 135–143.

[21] V. Gaede and O. Günther, "Multidimensional access methods," *ACM Computing Surveys*, vol. 30, no. 2, pp. 170–231, 1998.

[22] P. Gupta, R. Janardan, and M. Smid, "Computational geometry: generalized intersection searching," in *Handbook of Data Structures and Applications*. Chapman & Hall/CRC, 2005, ch. 64, pp. 1–17.

[23] J. M. Hellerstein, E. Koutsoupias, D. P. Miranker, C. H. Papadimitriou, and V. Samoladas, "On a model of indexability and its bounds for range queries," *Journal of the ACM*, vol. 49, no. 1, pp. 35–55, 2002.

[24] E. M. McCreight, "Priority search trees," *SIAM Journal on Computing*, vol. 14, no. 2, pp. 257–276, 1985.

[25] Y. Nekrich, "A data structure for multi-dimensional range reporting," in *SCG '07: Proc. of the 23rd Annual Symposium on Computational Geometry*. ACM, 2007, pp. 344–353.

[26] M. Pătraşcu, "Unifying the landscape of cell-probe lower bounds," in *FOCS '08: Proc. of the 49th Annual Symposium on Foundations of Computer Science*, 2008, pp. 434–443.

[27] S. Subramanian and S. Ramaswamy, "The P-range tree: a new data structure for range searching in secondary memory," in *SODA '95: Proc. of the 6th Annual Symposium on Discrete Algorithms*, 1995, pp. 378–387.

[28] R. E. Tarjan, "A class of algorithms that require nonlinear time to maintain disjoint sets," *Journal of Computer and System Sciences*, vol. 18, pp. 110–127, 1979.

[29] D. E. Vengroff and J. S. Vitter, "Efficient 3-D range searching in external memory," in *STOC '96: Proc. of the 28th annual ACM symposium on Theory of computing*, 1996, pp. 192–201.