New Data Structures for Orthogonal Range Searching

Stephen Alstrup*

The IT University of Copenhagen
Glentevej 67, DK-2400, Denmark
stephen@it-c.dk

Gerth Stølting Brodal†

BRICS ‡
Department of Computer Science
University of Aarhus
Ny Munkegade, DK-8000 Århus C, Denmark
gerth@brics.dk

Theis Rauhe§

The IT University of Copenhagen
Glentevej 67, DK-2400, Denmark
theis@it-c.dk

5th April 2001

Abstract

We present new general techniques for static orthogonal range searching problems in two and higher dimensions. For the general range reporting problem in $\mathbb{R}^3$, we achieve query time $O(\log n + k)$ using space $O(n \log^{1+\varepsilon} n)$, where n denotes the number of stored points and k the number of points to be reported. For the range reporting problem on an $n \times n$ grid, we achieve query time $O(\log\log n + k)$ using space $O(n \log^{\varepsilon} n)$. For the two-dimensional semi-group range sum problem we achieve query time $O(\log n)$ using space $O(n \log n)$.

*Partially supported by a grant from Direktør Ib Henriksens Fond.

†Partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

‡Basic Research in Computer Science, Center of the Danish National Research Foundation.

§Partially supported by a grant from Direktør Ib Henriksens Fond. Part of the work was done while the author was at BRICS.

1 Introduction

Let P be a finite set of points in $\mathbb{R}^d$ and Q a query range in $\mathbb{R}^d$. Range searching is the problem of answering various types of queries about the set of points which are contained within the query range, *i.e.,* the point set $P \cap Q$. A query is, *e.g.* to report the point set $P \cap Q$ (reporting queries), its cardinality $|P \cap Q|$ (counting queries), or simply to decide if $P \cap Q = \emptyset$ (emptiness queries). Orthogonal range searching is the special case where the query ranges are d-dimensional rectangles $[a_1, b_1] \times \cdots \times [a_d, b_d] \subseteq \mathbb{R}^d$.

Points can, *e.g.* represent a population of persons associated with a key with his or her age, sex, weight, salary etc. A typical orthogonal range query is of the form "find all males of age between 30 and 40 years with an income between \$20,000 and \$40,000".

The orthogonal range searching problem has numerous applications and has been studied extensively for the last decades, see *e.g.* [1, 3, 5, 6, 7, 8, 9, 10, 11, 13, 14, 16, 17, 20, 22, 24, 25, 26, 27, 30, 31, 40, 41, 42, 43, 45, 46, 47]. Willard [43] gives a comprehensive list of references on the subject and gives applications to the theory of databases. For surveys see, *e.g.* the survey by Agarwal [1], and the books by Mehlhorn [27] and Preparate and Shamos [31].

In this paper we consider various orthogonal range searching problems on static point sets. We give new techniques for static orthogonal range searching problems improving the previous best results [11, 14, 18, 30, 32, 41, 42] for various models, problems and dimensions: general range reporting in $\mathbb{R}^d$, for fixed $d \geq 3$, two-dimensional range reporting in rank space, and for the two-dimensional semi-group range sum problem. In the following we let n denote the number of stored points and k the number of points to be reported by a reporting query.

The model of computation we assume is a unit-cost RAM with word size logarithmic in n, as used for the most upper bounds, *e.g.* as in [11, 14, 18, 30, 32, 41, 42]. The remaining of the introduction gives a detailed discussion of our results.

1.1 Range reporting

Given n points $P \subseteq \mathbb{R}^d$, the general *static d-dimensional range reporting problem* is to construct a data structure for P that supports the query: **report**$([a_1, b_1] \times \cdots \times [a_d, b_d])$ which reports the point set $\{ (v_1, \ldots, v_d) \in P : a_1 \leq v_1 \leq b_1, \ \ldots \ , a_d \leq v_d \leq b_d \}$. For three dimensions we obtain the following result.

Theorem 1 *For the static three-dimensional range reporting problem in $\mathbb{R}^3$ there exists a data structure supporting queries in time $\mathrm{O}(\log n + k)$ and requiring space $\mathrm{O}(n \log^{1+\varepsilon} n)$.*

Chazelle in 1986 [13] gave a data structure for three dimensions with query time $\mathrm{O}(\log^2 n + k)$ and using space $\mathrm{O}(n \log^2 n / \log\log n)$. Willard in 1992 [42] improved the query time of Chazelle by a factor $\mathrm{O}(\log\log n)$ using fusion trees. Overmars in 1988 [30] gave a data structure with query time $\mathrm{O}(\log n \log\log n + k)$ using space $\mathrm{O}(n \log^2 n)$. The query time of Overmars was improved by a factor $\mathrm{O}(\log\log n / \log^{**} n)$[1] by Subramanian and Ramaswamy in 1995 [32]. Using a factor $\mathrm{O}(\log n)$ more space, the query time can be reduced by a factor $\mathrm{O}(\log^{**} n)$ [11]. Chazelle [14] gives a series of results using less space, but queries using more time. Table 1 summarizes the bounds for range reporting in $\mathbb{R}^3$.

Our data structure improves all the above mentioned results.

[1] $\log^{**} n$ is the number of times to apply $\log^* n$ to get a constant, and $\log^* n$ is the number of times to apply $\log n$ to get a constant.

Query time	Space	Source
$O(\frac{\log^2 n}{\log\log n} + k)$	$O(n\frac{\log^2 n}{\log\log n})$	[42]
$O(\log n \log^{**} n + k)$	$O(n\log^2 n)$	[32]
$O(\log n + k)$	$O(n\log^3 n)$	[11]
$O(\log^2 n + k)$	$O(n\log^{1+\varepsilon} n)$	[14]
$O(\log n + k)$	$O(n\log^{1+\varepsilon} n)$	New

Table 1: Orthogonal range reporting in $\mathbb{R}^3$.

Accepting a penalty for each reported point Chazelle [14] gave a data structure with query time $O(\log^2 n + k\log\log(4n/k))$ and using space $O(n\log n\log\log n)$, or query time $O(\log^2 n + k\log^{\varepsilon}(2n/k))$ and using space $O(n\log n)$.

Using a method of Willard and Lueker [46], the above bounds can be extended, for any fixed d, to d-dimensional range reporting, for $d \geq 4$, with a penalty of a factor $O(\log^{d-3} n)$ in space and query time (excluding the term involving k). We show how the above bounds can be extended for any fixed d, to d-dimensional range reporting, for $d \geq 4$, with a penalty of a factor $O(\log^{d-3+\varepsilon} n)$ in space and $O((\frac{\log n}{\log\log n})^{d-3})$ in query time (excluding the term involving k).

Finally, accepting a penalty for each reported point, orthogonal range search in $\mathbb{R}^3$ can be solved with query time $O(\log n(\log\log n)^2 + k\log\log n)$ using space $O(n\log n\log\log n)$. This result is obtained by applying the method of Willard and Lueker to one of our results for two-dimensional range searching on a grid in Section 1.2, using standard range reducing technique [30], as described in Section 2.2.

1.2 Two-dimensional range reporting

For n points in two-dimensional rank space, *i.e.,* an $n \times n$ grid, we have the following result.

Theorem 2 *For the static two-dimensional range reporting problem on an $n \times n$ grid, there exist data structures supporting queries in time* $O(\log\log n + k)$ *and* $O((\log\log n)^2 + k\log\log n)$ *respectively and requiring space* $O(n\log^{\varepsilon} n)$ *and* $O(n\log\log n)$ *respectively, for any constant* $\varepsilon > 0$. *In both cases the preprocessing time is expected* $O(n\log n)$.

For n points on an $n \times n$ grid Overmars [30] achieves query time $O(\log\log n + k)$ using space $O(n\log n)$. Our Theorem 2 improves the result of Overmars, and settle an open problem he raises by asking "It is not clear whether $\Omega(n\log n)$ storage is necessary for obtaining efficient solutions on a grid".

Chazelle [14] provides the following series of results for the case of points in $\mathbb{R}^2$:

Query time	Space
$O(\log n + k)$	$O(n\log^{\varepsilon} n)$
$O(\log n + k\log\log(4n/k))$	$O(n\log\log n)$
$O(\log n + k\log^{\varepsilon}(2n/k))$	$O(n)$

Using the standard range reduction from $\mathbb{R}^2$ to rank space, it can be seen that we only match or almost match the results of Chazelle in $\mathbb{R}^2$.

Query time	Space	Source
$\mathrm{O}(\log^2 n)$	$\mathrm{O}(n \log^{\varepsilon} n)$	[14]
$\mathrm{O}(\log^2 n \log\log n)$	$\mathrm{O}(n \log\log n)$	[14]
$\mathrm{O}(\log^{2+\varepsilon} n)$	$\mathrm{O}(n)$	[14]
$\mathrm{O}(\alpha(n) \log n)$	$\mathrm{O}(n \log n)$	[18]
$\mathrm{O}(\log n)$	$\mathrm{O}(n \log n)$	New

Table 2: Bounds for the orthogonal semi-group range sum problem in $\mathbb{R}^2$.

Restricting the model to, *e.g.* a pointer machine [34], Chazelle [15] has shown that reporting in time $\mathrm{O}(\log n + k)$ requires space $\Omega(n \log n / \log\log n)$. This bound is matched by an optimal upper bound for the pointer machine model [13].

1.3 The semi-group range sum problem

For the semi-group range sum problem we consider a commutative semi-group $\langle G, \oplus \rangle$, *i.e.,* we do not assume the elements in G to have additive inverses. Let P be n points in some space, *e.g.* $\mathbb{R}^2$, where each point $p \in P$ is associated with a semi-group element $e(p) \in G$. The semi-group range sum problem is to construct a data structure that for a given rectangular query range Q supports the query
sum(Q) returning the semi-group sum $\sum_{p \in P \cap Q} e(p)$.

A data structure cannot make use of subtractions or any internal property of the semi-group. Hence, a data structure for the semi-group range sum problem can be applied to any concrete choice of a semi-group which, *e.g.* may be available through function calls. For the semi-group range sum problem in $\mathbb{R}^2$ we have the following result.

Theorem 3 *For the static semi-group range sum problem in $\mathbb{R}^2$, there exists a data structure supporting queries in time $\mathrm{O}(\log n)$ and requiring space $\mathrm{O}(n \log n)$. The preprocessing time is expected $\mathrm{O}(n \log^2 n)$.*

Results for the range sum problem often only state the space used as the number of semi-group elements stored. However, for our data structure both the number of semi-group elements stored and the additional space required is $\mathrm{O}(n \log n)$. For the one-dimensional semi-group range sum problem in $[n]$, *i.e.,* a table of semi-group elements, Yao [47, 48] showed that using space m, the query time is $\Theta(\alpha(m, n) + n/(m - n + 1))$ in the arithmetic model [47, 48], where α is the functional inverse of Ackermann's function defined by Tarjan [33]. Chazelle and Rosenberg [18] showed how to achieve the upper bound on the RAM. Chazelle and Rosenberg combine their result for one dimension with the technique of Lueker and Willard [46] to get a result for two dimensions: Using space $\mathrm{O}(n \log n)$, queries can be answered in time $\mathrm{O}(\alpha(n) \log n)$. Chazelle [14] gives a series of results using less space, but queries using more time.

Table 2 summarizes the bounds for the orthogonal semi-group range sum problem in $\mathbb{R}^2$. The space bounds are the number of semi-group elements stored.

Willard [40] studied the range sum problem in the group model, *i.e.,* he makes use of the presence of additive inverses. Willard obtained query time $\mathrm{O}(\log n)$ using space $\mathrm{O}(n \log n)$, *i.e.,* the same bounds as we obtain for the semi-group model.

If we consider the product of query time and space, our result is the first that achieves a product trade-off of $O(n \log^2 n)$. Chazelle in [16] provides the following lower bound for the d-dimensional semi-group range sum problem: using m units of storage, the query time is $\Omega((\log n/ \log(2m/n))^{d-1})$ (see also Yao [48]). The lower bound is given for the dominance problem, *i.e.,* the special case where the ranges are of the form $[-\infty, b_1] \times \cdots \times [-\infty, b_d]$, and clearly holds for general orthogonal range sum queries. For the dominance problem Chazelle gives matching upper bounds in the arithmetic model for $m = \Omega(n \log^{1+\varepsilon} n)$. Lower bounds for the off-line version are given in [12].

2 Preliminaries

Let $[n]$ denote the set of integers $\{0, 1, \ldots, n-1\}$. We let $[a, b]$, denote the set (interval) of integers between a and b including a and b. The sets (intervals) $]a, b]$, $[a, b[$ and $]a, b[$ denotes the same set of integers but excluding a, excluding b, and excluding both a and b respectively. For $a > b$, the interval $[a, b] = \emptyset$ and for $a = b$, $[a, b[=]a, b] =]a, b[= \emptyset$. A rectangle is the cross product of two intervals. Let $u \geq 1$ denote an integer. Let $R \subseteq [u] \times [u]$ denote a rectangle and let S denote a set of points in $[u] \times [u]$. We let $rect(S, R)$ denote the set of points from S within rectangle R, *i.e.,* $rect(S, R) = S \cap R$. Finally, for an interval $I \subseteq [u]$ we let $rect_x(S, I)$ denote the set $rect(S, I \times [u])$ and $rect_y(S, I)$ the set $rect(S, [u] \times I)$.

2.1 Three-sided queries reporting

In our solutions for answering general queries in two dimensions, we will use data structures for *three-sided* queries in two dimensions. Let P be a point set in a two-dimensional space.

A three-sided query takes as arguments three coordinates x_1, x_2, y_1 and reports:

$$\textbf{report}(x_1, x_2, y_1)\textbf{:}\ \text{report}\ \{\, (x, y) \in P \,:\, x_1 \leq x \leq x_2 \ \wedge\ y \leq y_1 \,\}\ .$$

Note that the three-sided query equals the general query $\textbf{report}([x_1, x_2] \times [-\infty, y_1])$.

Fries *et al.* [23] considered three-sided queries in $[N] \times \mathbb{R}$. Given n lexicographically sorted points from $[N] \times \mathbb{R}$, they showed how to achieve query time $O(\log \log n + k)$, using $O(N + n)$ space and preprocessing time. We have the following result (following immediately from [24]):

Theorem 4 *For n points in $[N] \times \mathbb{R}$, using $O(N+n)$ space and preprocessing time, three-sided queries can be answered in $O(k)$ time.*

To show this let $S_x = \{y \mid (x, y) \in P\} \cup \{\infty\}$ be stored as sorted lists and let $s_x = \min S_x$, for $x \in [N]$. From [24, Sect. 3] we have that using $O(N+n)$ space and preprocessing time, we can for a query $\textbf{report}(x_1, x_2, y_1)$ in constant time find i such that $s_i = \min\{s_x \mid x_1 \leq x \leq x_2\}$. If $s_i > y_1$ we stop; otherwise we return the points $\{(i, y) \mid y \in S_i \ \wedge\ y \leq y_1\}$, and proceed with $\textbf{report}(x_1, i-1, y_1)$ and $\textbf{report}(i+1, x_2, y_1)$. In total we spend $O(k)$ time.

Corollary 1 *For n points in $[u] \times [u]$, using $O(n)$ space and expected preprocessing time, three-sided queries can be answered in $O(k + \log \log u)$ time.*

2.2 Reduction to rank space

Using a standard technique from the literature, *e.g.* Chazelle [14] and Gabow *et al.* [24], we can reduce a general static range searching problem in $\mathbb{R}^d$ to a range searching problem in the d-dimensional grid $[0, 1, \ldots, n-1]^d = [n]^d$, in the following denoted *rank space*. For a point x, let x_i denote the i^{th} coordinate of x. If P is a set of n points in $\mathbb{R}^d$, then P is translated to the set $\hat{P}$ in rank space by the order preserving mapping τ_P, where τ_P is defined by $(\tau_P(p))_i = \text{rank}(p_i, P_i)$, and $P_i = \{q_i \mid q \in P\}$ and $\text{rank}(x, S) = |\{y \in S \mid y < x\}|$. The construction of the set $\hat{P}$ is easily accomplished by sorting the points in P once with respect to each of the d coordinates. A d-dimensional range query $R = [a_1, b_1] \times \cdots \times [a_d, b_d]$ in $\mathbb{R}^d$ is translated to the range query $\tau_P(R) = [\hat{a}_1, \hat{b}_1] \times \cdots \times [\hat{a}_d, \hat{b}_d]$ in rank space by performing $2d$ binary searches such that $\hat{a}_i = \text{rank}(a_i, P_i)$ and $\hat{b}_i = \text{rank}(\text{pred}(b_i, P_i), P_i)$, where $\text{pred}(x, S) = \max\{y \in S \cup \{-\infty\} \mid y \leq x\}$. This translation satisfies $\tau_P(R \cap P) = \tau_P(R) \cap \hat{P}$. In the following we let the mapping τ_P from P to $\hat{P}$ be denoted the *range reduction* for P, and $\hat{P}$ the *range reduced* set of points. Note that $\hat{P}_i = [n_i]$, where n_i is the number of different i^{th}-coordinates of the points in P.

Algorithms given in this paper uses range reductions to reduce the problems defined for general spaces $\mathbb{R}^d$ to point sets in rank spaces. In order to support the range reduction from $\mathbb{R}^d$ to rank space, we sort the set of stored points by each of the d dimensions in time $\mathrm{O}(dn \log n)$. A range reduced query originally from $\mathbb{R}^d$ is then transformed to rank space using a binary search for each dimension in time $\mathrm{O}(d \log n)$. If the coordinates of the points are integers in a universe of size U, we can alternatively use the data structure of van Emde Boas that supports searches in time $\mathrm{O}(\log \log U)$ [28, 37, 38, 39] and uses space $\mathrm{O}(n)$. Depending on the computational model and the sort of problem, several different constructions can be used see *e.g.* [2, 4, 35, 36, 44].

3 Range searching on the grid

In this section we describe the data structure for range reporting on the two-dimensional grid. Let M be the input set of n points in $[n] \times [n]$. We assume word size of at least $\log n$, and when we say space cost, we measure this in terms of number of words used.

Our data structure uses the divide and conquer approach, and consists of a number of recursive levels. Each recursive level holds a number of auxiliary range searching structures supporting various limited kinds of range queries. We consider a division of the points into subsets called rows and columns. Each point is represented in at most two recursive substructures for a row subset and a column subset.

In the following we use the function $f(m) = \lceil \sqrt{m \log m} \rceil$. We let $f^{(0)}(m) = m$ and $f^{(k)}(m) = f(f^{(k-1)}(m))$ for $k \geq 1$. We let $f^*(m)$ denote the minimal integer k such that $f^{(k)}(m) \leq 3$. We need the following fact for f.

Fact 1 *For any integer l, $1 \leq l \leq f^*(n)$, $2^l \log(f^{(l)}(n)) \leq 4 \log n$ and $f^*(n) = \log \log n + \mathrm{O}(1)$.*

Consider a point set S from a universe $[u] \times [u]$, where $u \leq n$. Let $m \leq n$ be the size of S. We define the row and column subsets relative to a set of row borders $\mathcal{R} \subseteq [u+1]$ and column borders $\mathcal{C} \subseteq [u+1]$ defined as follows. Let $c(0) = 0$. The k^{th} column border $c(k)$ is defined to be the minimal $x > c(k-1)$ such that $|rect_x(S, [c(k-1), x])| > f(m)$. If no such x exists, $c(k) = u$ and is then the last column border. The k^{th} *column* is the set $C(k) = rect_x(S, [c(k-1), c(k)[)$

and the *interior* of the column is $\dot{C}(k) = rect_x(S,]c(k-1), c(k)[)$. Note that by definition of the borders, $|\dot{C}(k)| \leq f(m)$ for all k. The row borders and associated rows are defined similarly. That is $r(0) = 0$ and $r(k) = \min(\{ y > r(k-1) : |rect_y(S, [r(k-1), y])| > f(m) \} \cup \{u\})$. The k^{th} row is $R(k) = rect_y(S, [r(k-1), r(k)[)$ and $\dot{R}(k) = rect_y(S,]r(k-1), r(k)[)$. Finally, let $Q(i,j)$ denote the intersection of column i and row j, *i.e.,* $Q(i,j) = C(i) \cap R(j) = rect(S, [c(i-1), c(i)[\times [r(j-1), r(j)[)$. By definition of row and column borders, there can be at most $\lceil 2m/f(m) \rceil$ columns and $\lceil 2m/f(m) \rceil$ rows. We define the *top set* of points $\hat{S} \subseteq [\lceil 2m/f(m) \rceil]^2$ by $(i,j) \in \hat{S}$ if and only if $Q(i,j) \neq \emptyset$.

A range query for the query rectangle $[a,b] \times [c,d] \subseteq [u] \times [u]$ can be expressed in terms of range queries for the above sets. We split between two cases for the query.

Case a) $[a,b] \cap \mathcal{C} \neq \emptyset$ and $[c,d] \cap \mathcal{R} \neq \emptyset$.
In this case let $[i_a, i_b]$ be the set of column borders spanned by $[a,b]$, *i.e.,* $\{c(i_a), c(i_a+1), \ldots, c(i_b)\} = [a,b] \cap \mathcal{C}$. Similarly let $[i_c, i_d]$ be the set of row borders spanned by $[c,d]$. Define the rectangles $C_l = [a, c(i_a)[\times [c,d]$, $C_r = [c(i_b), b] \times [c,d]$, $R_{\text{low}} = [c(i_a), c(i_b)[\times [c, r(i_c)[$ and $R_{\text{up}} = [c(i_a), c(i_b)[\times [r(i_d), d]$. Let $\hat{R} = rect(\hat{S},]i_a, i_b] \times]i_c, i_d])$. Then $rect(S, [a,b] \times [c,d])$ can be expressed as the following *disjoint* union

$$\begin{aligned} &\Big(\bigcup_{(i,j) \in \hat{R}} Q(i,j) \Big) \cup \\ &rect(C(i_a), C_l) \cup rect(C(i_b+1), C_r) \cup \\ &rect(R(i_c), R_{\text{low}}) \cup rect(R(i_d+1), R_{\text{up}}). \end{aligned} \tag{1}$$

Case b) $[a,b] \cap \mathcal{C} = \emptyset$ or $[c,d] \cap \mathcal{R} = \emptyset$.
In this case the query rectangle $[a,b] \times [c,d]$ is completely within the interior of a row or a column. That is, if it is completely within a column (in case of $[a,b] \cap \mathcal{C} = \emptyset$) we can express $rect(S, [a,b] \times [c,d])$ as $rect(\dot{C}(k), [a,b] \times [c,d])$ for the unique k satisfying $c(k-1) < a \leq b < c(k)$. Similar for the rectangle completely within a row (in case of $[c,d] \cap \mathcal{R} = \emptyset$), $rect(S, [a,b] \times [c,d]) = rect(\dot{R}(k), [a,b] \times [c,d])$ for the unique k satisfying $r(k-1) < c \leq d < r(k)$.

Hence by the above two cases, we can answer any reporting query for a rectangle provided access to reporting queries for the following types of ranges.

1. Three-sided rectangle ranges contained in a column or a row, with a side fixed to a column or row border, *i.e.,* as the rectangle ranges C_l, C_r, R_{low} and R_{up} in (1).

2. A range properly included in a column or row as in Case b).

3. A general range query for the top set $\hat{S}$ within domain $[\lceil 2m/f(m) \rceil]^2$. This range query computes the points corresponding to $\hat{R}$ in (1). Using these points and information stored for each $Q(i,j)$ for a $(i,j) \in \hat{S}$, we can form the union corresponding to the first term in (1).

Our data structure reflects the above partition into rows and columns, with auxiliary structures supporting queries 1. and 3. above, and recursive structures for 2. To be more precise, the data structure consists of ℓ recursive levels (which by Fact 1 will turn out to be $\log \log n + \mathrm{O}(1)$), starting with the input set at level 0. Consider a structure at level l,

$0 \le l \le \ell$, storing point set S. If S has size less than a constant larger than 3, the recursion stops and we represent the points in a list with queries supported by a linear scan in constant time. Otherwise S is partitioned according to the above description. For each $1 < k \le |\mathcal{C}|$, the column border $c(k)$ is associated two three-sided range searching structures, a structure for points in $C(k-1)$ and one for all points in $C(k)$. Both supports the three-sided queries with a side fixed to $c(k)$. That is, these queries enable answers for the rectangles as given in 1. For $k = 1$ or $k = |\mathcal{C}|$, the border $c(k)$ is only associated one structure for the points $C(k)$ and $C(k-1)$ respectively. Similarly we associate three-sided structures for the rows with a side fixed to a row border for rectangles like R_{low} and R_{up} in Case a).

The point set $\hat{S}$ is represented in an general range searching structure we call the top structure. In addition to this, we store information for each set $Q(i,j)$, *i.e.,* a list of the points. For each $Q(i,j)$ we keep this information in an entry in a two-dimensional array with $\lceil 2m/f(m) \rceil^2$ entries. Hence in order to report the points for $Q(i,j)$ we simply return the list for entry (i,j) in the array.

Finally, for each interior point set I, *i.e.,* I is a set $\dot{C}(k)$ and $\dot{R}(k)$ for an integer k, we store I in a recursive structure corresponding to level $l+1$. We will use two strategies for this recursive representation depending upon the time and space cost we aim at. The recursive point set for I may either be stored relative to the universe used for S, or we may reduce the universe to rank space for I. In the latter case, all structures recursively represented at level $l+1$ are range reduced, and we say a range reduction on level l takes place. For levels where a range reduction takes place, we keep a van Emde Boas data structure which enable us to transform a rectangle query within a range reduced point set I to the rank space domain for I. Furthermore we also store a perfect hash table [21] enabling us to map the points from range reduced points sets at level $l+1$ back to the original domain for S. We call the non-recursive data structures associated the recursive structures at level l such as the three-sided range searching structures and the top structure for auxiliary structures at level l.

The set of recursive range searching data structures at level l is denoted $D(l)$, for instance $D(0)$ is the general structure for the input set M, $D(1)$ is the set of structures for the interior column and row sets for M. Let $d \in D(l)$ be a recursive structure at level l. The number of points stored in d is denoted $m(d)$. We let $u(d)$ denote the universe size for structure d, *i.e.,* d stores points in the grid $[u(d)] \times [u(d)]$. We let $u(l)$ be the size of the largest universe size at a recursive level l, *i.e.,* $u(l) = \max_{d \in D(l)} u(d)$. Before describing the query computation we state three simple lemmas relevant for the analysis of the data structure.

Lemma 1 *For any l, $0 \le l \le \ell$, the number of points $m(d)$ in a structure $d \in D(l)$ is bounded by $f^{(l)}(n)$.*

Proof. Proof by induction on l, using that the largest structure in $D(l+1)$ contains at most $f(m)$ points, where $m = \max_{d \in D(l)} m(d)$, and the initial level 0 has n points. ∎

Lemma 2 *The number of levels ℓ is bounded by $\mathrm{O}(\log\log n)$.*

Proof. By Lemma 1, any structure at level $f^*(n) = \mathrm{O}(\log\log n)$ contains at most 3 points and hence the recursion can not have depth of more than this. ∎

Lemma 3 *For any level l, $0 \le l \le \ell$, $\sum_{d \in D(l)} m(d) \le 2^l n$.*

Proof. Proof by induction on l. At the initial level 0 it clearly holds. Next for a point set S at a level l, a point in S is represented in at most two recursive structures at level $l+1$, *i.e.,* in the interior point sets for a column and a row. Hence the number of points at level $l+1$ is at most the double of level l. ∎

3.1 Query computation

Consider a query for rectangle $[a, b] \times [c, d]$ to a structure $d \in D(l)$ for some level l, where $0 \le l \le \ell$. First the computation decides whether the query rectangle satisfy Case a) or Case b) above. That is, we need a structure to decide whether the interval $[a, b]$ contains a column border or whether $[c, d]$ contains a row border. In case one of these intervals does not contain a border, the structure returns the column (or row) number the rectangle is contained in, *i.e.,* the predecessor for a (or c). We call the data structure supporting this kind of query for column or row borders, the interval range (IR) structure. We will express the time cost of a query to the IR structure in terms of n and let $q(n)$ denote this cost. The IR structures we use will have linear space cost in terms of $m(d)$, *i.e.,* a bit cost of $O(m(d) \log u(l))$. If the query is in Case a), there will be no further recursive calls, and the computation is reflected by the expression (1). That is the computation consists of computing the five rectangles R_{up}, R_{low}, C_l, C_r and $\hat{R}$ using the respective three-sided column and row range searching structures and top structure. In total, the computation of the rectangle boundaries takes time $O(\log \log n)$ using the van Emde Boas data structure. Points to be returned are then the collection of points from these queries together with the points in the lists associated sets $Q(i, j)$, for points (i, j) to be reported for $\hat{R}$ in (1).

In Case b) we need a recursive call for an interior point set I for either a column or a row. If a range reduction takes place, *i.e.,* I is represented recursively in rank space, we need to transform the query rectangle $[a, b] \times [c, d]$ to the corresponding range reduced region valid for the rank space for I at level $l+1$. Furthermore, each recursively returned point from this query needs to be mapped back to the domain for this level l using the perfect hash table storing the inverse of the range reduction for I.

We summarize the time cost of the various steps for a level l. The first step is to establish whether the query corresponds to Case a) or Case b). We can use a van Emde Boas data structure in which case we get $q(n) = O(\log \log n)$. Later we will describe how to avoid this cost of $O(\log \log n)$, using a certain approximate version of IR structure allowing time cost of $q(n) = O(1)$ within bit cost $O(m(d) \log u(l))$.

Next consider the computation needed for Case b). The IR structure gives us the number of the column or row containing the rectangle, *i.e.,* which recursive structure to call. If a range reduction takes place at l, there is an additional cost of $O(\log \log n + k)$; we need to compute the range reduced query rectangle for the recursive call appropriate for the domain at level $l+1$. In addition we need $O(k)$ calls to the perfect hash table for mapping the points from the domain of level $l+1$ back to original domain at level l.

Next consider Case a). First we need $O(\log \log n)$ time for the computation of the query boundaries. By Corollary 1 these three-sided queries take time $O(\log \log u(l) + k) = O(\log \log n + k)$ for k elements to be reported. Hence the overall work for Case a) is $O(\log \log n + k)$.

Let r denote the number of levels for which a range reduction takes place. For a query computation, the total time spend on recursive levels of this kind is by the above analysis $\mathrm{O}(r \log\log n + rk)$. For levels without a range reduction and where a Case b) computation takes place we only use constant time. Finally, since there is only one level for which a query computation corresponds to Case a) (we do not recurse from such case) and this takes time $\mathrm{O}(\log\log n + k)$, the total computation for all levels along the computation path is

$$\begin{aligned} & \mathrm{O}(\ell q(n)) + \mathrm{O}(\log\log n + k) + \mathrm{O}(r \log\log n + rk) \\ & = \mathrm{O}(q(n)\log\log n + (r+1)\log\log n + (r+1)k). \end{aligned} \tag{2}$$

also using Lemma 2.

3.2 Analysis

In our analysis we will bound the number of bits used at each recursive level. First we analyze the number of bits used for auxiliary structures associated a recursive level l, in terms of $u(l)$ and n.

By Lemma 3 there is a total of $2^l n$ points at level l. Since the IR structure uses linear space and by Corollary 1 each auxiliary three-sided structure storing m points uses at most $\mathrm{O}(m)$ words of size $\mathrm{O}(\log u(l))$, the total bit cost of these structures are bounded by $\mathrm{O}(2^l n \log u(l))$.

Next, each *top structure* associated a structure with a total of m points, keeps at most $s = (\lceil 2m/f(m) \rceil)^2 = \mathrm{O}(m/\log m)$ points. Hence, any auxiliary general range searching structure using $\mathrm{O}(s \log s)$ words of size $\log u(l)$ for s points in universe $[u(l)]^2$, keeps the total bit cost for the top structure at $\mathrm{O}(m \log u(l))$. Hence, the total bit cost of top structures at level l is bounded by $\mathrm{O}(2^l n \log u(l))$. The bound on $\mathrm{O}(s \log s)$ for space cost is met by the Overmars' data structure [30] with queries in time $\mathrm{O}(\log\log u(l) + k) = \mathrm{O}(\log\log n + k)$.

For the reporting case we consider two variants of the above structure with different trade-offs between space and time. We start by the variant with the best space cost.

In this variant a range reduction takes place on each recursive level. Hence the universe size of a point set at level l is bounded by the number of points in the set. Since range reduction takes place at level $l-1$, $u(l)$ is bounded by $\max_{d \in D(l)} m(d) \leq f^{(l)}(n)$, the last inequality follows from Lemma 1. Hence the total bit cost of structures at level l is $\mathrm{O}(n 2^l \log(f^{(l)}(n)))$.

Fact 1 bound the total bit cost for all levels by

$$n \log n + \sum_{l=1}^{\ell} n 2^l \log(f^{(l)}(n)) = \mathrm{O}(n f^*(n) \log n) = \mathrm{O}(n \log n \log\log n) \ .$$

Hence, this variant has the claimed $\mathrm{O}(n \log\log n)$ bound on space cost in terms of words of size $\Omega(\log n)$.

In this variant we use the van Emde Boas data structure for the IR structure, *i.e.,* $q(n) = \mathrm{O}(\log\log n)$ and hence by the bound given in (2) we get a total time cost of $\mathrm{O}((\log\log n)^2 + k \log\log n)$ since $r = \ell = \mathrm{O}(\log\log n)$. This proves the second part of Theorem 2.

The other trade-off variant we consider for reporting uses more space, but provides better query performance. Let c be an integer such that $1 \leq c \leq \ell$. The only difference from the previous variant is that range reductions only take place on levels $l-1$ where c divides l. This leads to an increase of space cost which is analyzed as follows. Let l be a recursive level for which c divides l. Since a range reduction takes place on level $l-1$, $u(l)$ is bounded by $f^{(l)}(n)$. Using the same argumentation as for the previous reporting variant, the bit cost for any level l', $l \leq l' < l+c$ is bounded by $2^{l'} n \log(u(l'))$. Since $u(l') = u(l)$ we get $2^{l'} n \log(u(l')) \leq 2^{l'} n \log(f^{(l)}(n))$. From this we can bound the sum of bit costs of recursive levels $l, l+1, \ldots, l+c-1$ by $\sum_{l'=l}^{l+c-1} 2^{l'} n \log(f^{(l)}(n)) \leq n 2^c 2^l \log(f^{(l)}(n)) = \mathrm{O}(2^c n \log n)$, since $2^l \log(f^{(l)}(n))$ is $\mathrm{O}(\log n)$ by Fact 1.

Hence for any constant $\epsilon > 0$, we can choose another constant $\epsilon' > 0$ where we for $c = \lceil \epsilon' \ell \rceil$ obtain a total bit cost of $\mathrm{O}((\ell/c) 2^c n \log n) = \mathrm{O}(n \log^{1+\epsilon} n)$. In terms of $\log n$ size words, the space cost is thus $\mathrm{O}(n \log^{\epsilon} n)$. Furthermore, the number r of levels where a range reduction takes place is $\mathrm{O}(\ell/c) = \mathrm{O}(1/\epsilon') = \mathrm{O}(1)$. Thus the total time cost is $\mathrm{O}(q(n) \log\log n + (\log\log n)(\ell/c+1) + k(\ell/c+1)) = \mathrm{O}(q(n) \log\log n + k)$ by the general bound (2) on the time cost for a query. Implementing the IR structure using the van Emde Boas data structure, leads to $\mathrm{O}((\log\log n)^2 + k)$.

In order to avoid the $\log\log n$ factor for the additive term we use a modified version of a data structure of Miltersen *et al.* [29] which supports range reporting in one dimension in constant time per point to be reported. By a lookup of an associated column and row number for a returned point, we can determine the information needed for the IR query. Unfortunately the data structure of Miltersen *et al.* [29] uses slightly too much space, *i.e.*, it has a bit cost of $\mathrm{O}(m(d) \log^2 u(l))$. By only keeping a sparse sample of $m(d)/\log u(l)$ points in the structure, we can reduce the bit cost to the desired $\mathrm{O}(m(d) \log u(l))$. However, this sparsification leads to a special case for interval ranges that contain very few points (less than $\log u(l)$). This case can be handled by a simple additional linear space auxiliary structure for thin rows and thin columns with at most $\log(u(l))$ points in each. For these thin rows and columns we can support general range queries in time $\mathrm{O}(\log\log n + k)$ for k points to be reported. Details will be given in the full paper. We conclude that this variant of the range reporting proves the first part of Theorem 2.

4 Semi-group sum

In the semi-group variant we measure the space cost in terms of the number of stored semi-group sums and the bit cost of remaining parts of the data structure. We will analyze these two measures separately in what follows. The structure is very similar to the reporting variants, and we will thus just describe how to modify these variants to obtain the result.

Before we begin a description we need a certain parameterized version of the semi-group variant for three-sided and general range searching structures.

Lemma 4 *Let $S \subseteq [n] \times [n]$ be a set of points with $n = |S|$. For any integer parameter p, $1 \leq p \leq \log n$ we can construct a three-sided range searching data structure for S in time $\mathrm{O}(n \log n)$ time using space $\mathrm{O}(n \log n)$ containing at most $\mathrm{O}(n \log n/p)$ semi-group elements, such that range queries can be answered in $\mathrm{O}(p \log n)$ time.*

For the proof see Section 5.

In addition to the three-sided range query we also need a general semi-group range searching data structure, but for this we allow a $\log n$ factor extra in the space cost, stated in the following lemma.

Lemma 5 *Let p be an integer parameter such that $1 \leq p \leq \log n$. The general semi-group sum problem can be solved in time $\mathrm{O}(p \log n)$ and space $\mathrm{O}(n \log^2 n/p)$ in terms of semi-group elements, and space $\mathrm{O}(n \log^2 n)$ in terms of words.*

For the proof see Section 5.

The semi-group variant is a modification of the data structure from Section 3. Fix c such that $1 < c \leq 2$. For the three-sided range searching structures at level l we use the structure from Lemma 4 with parameter $p = \lceil c^l \rceil$. For the top structure we use the data structure from Lemma 5 again with parameter $p = \lceil c^l \rceil$. The semi-group element $e(i,j)$ associated $(i,j) \in \hat{S}$ is the semi-group sum of elements in $Q(i,j)$. The analogy of (1) for the semi-group sum is then

$$\begin{aligned}
&\mathit{rect}(S, [a,b] \times [c,d]) = \sum_{(i,j) \in \hat{R}} e(i,j) \oplus \\
&\mathbf{sum}(\mathit{rect}(C(i_a), C_l)) \oplus \mathbf{sum}(\mathit{rect}(C(i_b+1), C_r)) \oplus \\
&\mathbf{sum}(\mathit{rect}(R(i_c), R_{\text{low}}) \oplus \mathbf{sum}(\mathit{rect}(R(i_d+1), R_{\text{up}})).
\end{aligned}$$

Note that the term $\sum_{(i,j) \in \hat{R}} e(i,j)$ corresponds to a single range query for the top set $\hat{S}$ with the associated semi-group sums $e(i,j)$ for the points $(i,j) \in \hat{S}$.

4.1 Analysis

Consider a structure $d \in D(l)$ for a level l. We will show that the number of semi-group elements used for the the auxiliary three-sided structures and the top structure for d is bounded $\mathrm{O}((m(d) \log m(d))/c^l)$. First, the top structure contains at most $\mathrm{O}((m(d)/f(m(d)))^2) = \mathrm{O}(m(d)/\log m(d))$ points. Hence the use of structure from Lemma 5 with parameter $p = \lceil c^l \rceil$ implies a cost of $\mathrm{O}((m(d) \log m(d))/c^l)$ semi-group elements. For the three-sided structures we store at most $\mathrm{O}(m(d))$ points, hence by Lemma 4 also leading to a cost of $\mathrm{O}(m(d) \log m(d)/c^l)$ semi-group elements.

By these bounds and by Lemma 1, Lemma 3 and Fact 1 we have the following total bound on the number of semi-group elements stored in auxiliary structures at level l

$$\begin{aligned}
\sum_{d \in D(l)} (m(d) \log m(d))/c^l &\leq n 2^l \log f^{(l)}(n)/c^l \\
&= \mathrm{O}((n \log n)/c^l).
\end{aligned}$$

For the chosen parameter c, the sum of semi-group element cost of all levels is

$$\mathrm{O}(n \sum_{l=0}^{\ell} \log n/c^l) = \mathrm{O}(n \log n)$$

as desired. Using argumentation very similar to the reporting variant (except from an additional $\log n$ factor) the additional space cost (in terms of $\log n$ size words) is bounded by $\mathrm{O}(n \log n)$.

The time cost is also very similar to the analysis for the reporting variant. Time cost for traversing through the recursive structures until we reach a structure at some level l for which Case a) holds is the same, *i.e.,* with a cost of $\mathrm{O}(\ell \log\log n) = \mathrm{O}((\log\log n)^2)$. The remaining computation is the calls for three-sided queries and the top structure at the level l where the query satisfy Case a) for a structure $d \in D(l)$. By Lemma 1 $m(d)$ is bounded by $f^{(l)}(n)$. The chosen structures for the three-sided queries and the top structure has time cost bounded by $\mathrm{O}(c^l \log m(d))$ which by Fact 1 is $\mathrm{O}(\log n)$ since $c \leq 2$. Hence the total time cost is $\mathrm{O}(\log n)$ as claimed.

5 Range searching in d dimensions

Let A be a data structure for d-dimensional range reporting, using space $\mathrm{O}(s(n))$ and time $\mathrm{O}(t(n)+k)$, where $t(n) \geq \log\log n$, for n points $P \subseteq \mathbb{R}^d$. We show how to extend A to support $(d+1)$-dimensional range reporting, using space $\mathrm{O}(s(n)\log^{1+\varepsilon} n)$ and $\mathrm{O}(t(n)(\log n/\log\log n)+k)$ time. For a point x, let x_i denote the i^{th} coordinate of x. To avoid tedious details we assume $x_i \neq y_i$ for all $x, y \in P$ and i. Let $P_i = \{p_i | p \in P\}$. Let T be a rooted tree, with n leafs, node degree B, where the different between two leafs depth is at most 1. The leafs are ordered from left to right, and to the j^{th} leaf we associate the point $p \in P$, where $\mathrm{rank}(p_{d+1}, P_{d+1}) = j$. For a node in T, we for each of the B^2 pairs of its children $c_1, \ldots, c_B$ associate a d-dimensional data structure. Such data structure associated a pair, say (c_i, c_j), contains the points associated all leafs to descendents of $c_i, \ldots, c_j$. A point in these d-dimensional data structures is only represented by the first d coordinates. Each point is represented in B^2 d-dimensional data structures at each of the $\mathrm{O}(\log n/\log B)$ levels in the tree, leading to space cost $\mathrm{O}(s(n)B^2\log n/\log B)$. A $d+1$-dimensional query can now be answered by $\mathrm{O}(\log n/\log B)$ d-dimensional queries. In addition we need $\mathrm{O}(\log n/\log B\log B)$ time to determine these queries. Choosing $B = (\log n)^{1/l}$, for a constant $l > 1$, we have a $d+1$ dimensional data structure with the claimed complexity. The construction can be repeated a fixed number of times proving the following theorem.

Theorem 5 *Given a data structure A for two-dimensional static range reporting problem, using space $\mathrm{O}(s(n))$ and time $\mathrm{O}(t(n)+k)$, $t(n) \geq \log\log n$, for n points A can be extended, for any fixed $d \geq 3$, to d-dimensional range reporting, with space complexity $\mathrm{O}(s(n)\log^{d-2+\varepsilon} n)$ and time complexity $\mathrm{O}(k + t(n)(\frac{\log n}{\log\log n})^{d-2})$.*

Combining Theorem 5 with the first part of Theorem 2, using the observations in Section 2.2, proves Theorem 1.

6 Subproblems for the semi-group result

Proof. *(Lemma 4)* We solve the three-sided range query by using a dynamic to static transformation technique of a persistent dynamic one-dimensional version of the semi-group sum problem. The one-dimensional problem is as follows. For an A array of semi-group elements we will support two operations: $\mathbf{update}(i, f) : A[i] := A[i] \oplus f$ and $\mathbf{interval}(i, j) := A[i] \oplus A[i+1] \cdots \oplus A[j]$. Initially $A[i] = e$ for all i, where e is the neutral element for the semi-group. Over the array we span a complete binary tree (we assume without loss of generality that $n = 2^k$ for integer $k \geq 0$). The leafs are ordered from left to right, and the ith leaf holds the value of $A[i]$. An internal node that spans the leafs from i to j holds the value $\mathbf{interval}(i, j)$. An $\mathbf{update}(i, f)$ operation corresponds to updating the ith leaf. Updating a leaf i is done by updating in constant time each of its $\mathrm{O}(\log |A|)$ ancestors. Similarly, a query can be answered by computing the sum of $\mathrm{O}(\log |A|)$ values from internal nodes. Now we use a standard approach to get a solution for the static three-sided two-dimensional range query. The degree of the nodes in the above structure is clearly bounded by a constant implying that we can use a persistent technique by Driscoll *et. al.* [19]. This is done with a worst case slowdown $\mathrm{O}(1)$ for queries, amortized slowdown $\mathrm{O}(1)$ for updates and amortized space cost $\mathrm{O}(1)$ per memory modification. Using the above one-dimensional structure for the first coordinate of the points, we now insert the points in increasing order by the second coordinate. A three-sided query (x_0, x_1, y) corresponding to $\mathbf{interval}(x_0, x_1)$ in the persistent one-dimensional structure at the time where all points with second coordinates $\leq y$ have been inserted, and no one else. This gives an $\mathrm{O}(n \log n)$ space and $\mathrm{O}(\log n)$ query time solution for three-sided range queries. Next we show how to decrease the space used for semi-group elements to $\mathrm{O}(n \log n/p)$, by increasing the query time to $\mathrm{O}(p \log n)$ still using $\mathrm{O}(n \log n)$ space in total. This is done by a slight modification of the above dynamic one-dimensional algorithm. For an internal node z spanning the leafs $i \cdots j$ we associate a bucket of pointers to semi-group elements. Updating z with f we insert a pointer to f in z's bucket. If the bucket holds p pointers, we empty the bucket and update z. Let z hold the semi-group value $v(z)$, let the semi-group element a pointer q points to be $v(q)$, and let the pointers in z's bucket be $q_1, \cdots q_p$. Updating z is to set $v(z) := v(z) \oplus v(q_1) \oplus \cdots \oplus v(q_p)$. This change does not decrease the total space used, but it does decrease the number of semi-group elements in memory to $\mathrm{O}(n \log n/p)$. To perform a query we have to examine $\mathrm{O}(\log n)$ ancestors and their buckets of size p, leading to complexity $\mathrm{O}(p \log n)$ for a query. ∎

Proof. (Lemma 5) To avoid tedious details we assume $x_1 \neq y_1$ for all $x, y \in S$. We divide the points into two subsets A and B, such that $A = \{\, p : rank(p_1, S_1) < |S|/2 \,\}$ and $B = S \setminus A$. To each of the point sets A and B we associate a three-sided structure that enable us to answer queries not entirely enclosed in either A or B. In order to answer queries entirely enclosed in either A or B we associate a recursive structure. Now using $\mathrm{O}(\log n)$ time we find the structure in which we can combine two three-sided queries for the answer leading to the complexity $\mathrm{O}(\log n + p \log n)$. Since a point is at most included in $\mathrm{O}(\log n)$ structures we also achieve the claimed space complexity. ∎

References

[1] P. Agarwal. Range searching. In *Handbook of Discrete and Computational Geometry, CRC Press*. 1997.

[2] A. Andersson, T. Hagerup, S. Nilsson, and R. Raman. Sorting in linear time? *Journal of Computer and System Sciences*, 57(1):74–93, Aug. 1998.

[3] Z. Aviad and E. Shamir. A direct dynamic solution to range search and related problems for product regions. In *22th Annual Symposium on Foundations of Computer Science*, pages 123–126, Los Alamitos, Ca., USA, Oct. 1981. IEEE Computer Society Press.

[4] P. Beame and F. E. Fich. Optimal bounds for the predecessor problem. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 295–304, New York, May 1–4 1999. ACM Press.

[5] J. L. Bentley. Decomposable searching problems. *Information Processing Letters*, 8(5):244–251, 1979.

[6] J. L. Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229, 1980.

[7] J. L. Bentley and J. H. Friedman. Data structures for range searching. *ACM Computing Surveys*, 11(4):397–409, 1979.

[8] J. L. Bentley and H. A. Maurer. Efficient worst-case data structures for range searching. *Acta Informatica*, 13(2):155–168, 1980.

[9] J. L. Bentley and M. I. Shamos. A problem in multivariate statistics: Algorithm, data structure, and applications. In *Proceedings of the Fifteenth Allerton Conference on Communication, Control, and Computing*, pages 193–201, 1977.

[10] A. Bolour. Optimal retrieval algorithms for small region queries. *SIAM Journal on Computing*, 10(4):721–741, 1981.

[11] P. Bozanis, N. Ktsios, C. Makris, and A. Tsakalidis. New results on intersection query problems. Unpublished manuscript, 1996.

[12] Chazelle. Lower bounds for off-line range searching. *Discrete and Computational Geometry*, 17, 1997.

[13] B. Chazelle. Filtering search: a new approach to query-answering. *SIAM Journal on Computing*, 15(3):703–724, 1986.

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

[15] B. Chazelle. Lower bounds for orthogonal range searching: I. the reporting case. *Journal of the Association for Computing Machinery*, 37(2):200–212, Apr. 1990.

[16] B. Chazelle. Lower bounds for orthogonal range searching: II. the arithmetic model. *J. ACM.*, 37(3):439–463, 1990.

[17] B. Chazelle and H. Edelsbrunner. Linear space data structures for two types of range search. *Discrete and Computational Geometry*, 2:113–126, 1987.

[18] B. Chazelle and B. Rosenberg. Computing partial sums in multidimensional arrays. In K. Mehlhorn, editor, *Proceedings of the 5th Annual Symposium on Computational Geometry (SCG '89)*, pages 131–139, Saarbrücken, FRG, 1989. ACM Press.

[19] J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. E. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38(1):86–124, 1989.

[20] R. A. Finkel and J. L. Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, 1974.

[21] M. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *Journal of the Association for Computing Machinery*, 31(3):538–544, 1984.

[22] M. L. Fredman. Lower bounds on the complexity of some optimal data structures. *SIAM Journal on Computing*, 10(1):1–10, 1981.

[23] O. Fries, K. Mehlhorn, S. Näher, and A. Tsakalidis. A $\log\log n$ data structure for three-sided range queries. *Information Processing Letters*, 25(4):269–273, June 1987.

[24] H. N. Gabow, J. L. Bentley, and R. E. Tarjan. Scaling and related techniques for geometry problems. In *Proc. of the Sixteenth Annual ACM Symposium on Theory of Computing*, pages 135–143, 1984.

[25] G. S. Lueker. A data structure for orthogonal range queries. In *19th Annual Symposium on Foundations of Computer Science*, pages 28–34, Long Beach, Ca., USA, Oct. 1978. IEEE Computer Society Press.

[26] E. M. McCreight. Priority search trees. *SIAM Journal on Computing*, 14(2):257–276, 1985.

[27] K. Mehlhorn. *Data Structures and Algorithms: 3. Multidimensional Searching and Computational Geometry*. Springer, 1984.

[28] K. Mehlhorn and S. Näher. Bounded ordered dictionaries in $O(\log\log N)$ time and $O(N)$ space. *Journal of the ACM*, 35(4):183–189, 1990.

[29] P. B. Miltersen, N. Nisan, S. Safra, and A. Wigderson. On data structures and asymmetric communication complexity. *Journal of Computer and System Sciences*, 57(1):37–49, 1998.

[30] M. H. Overmars. Efficient data structures for range searching on a grid. *Journal of Algorithms*, 9(2):254–275, 1988.

[31] F. P. Preparata and M. Shamos. *Computational Geometry*. Springer-Verlag, New York, 1985, 1985.

[32] S. Subramanian and S. Ramaswamy. The P-range tree: A new data structure for range searching in secondary memory. In *Proceedings of the 6th Annual Symposium on Discrete Algorithms*, pages 378–387. ACM Press, 1995.

[33] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, 1975.

[34] R. E. Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *Journal of Computer and System Sciences*, 18(2):110–127, Apr. 1979.

[35] M. Thorup. Randomized sorting in $O(n \log\log n)$ time and linear space using addition, shift, and bit-wise boolean operations. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 352–359, New Orleans, Louisiana, 5–7 Jan. 1997.

[36] M. Thorup. Faster deterministic sorting and priority queues in linear space. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 550–555, San Francisco, California, 25–27 Jan. 1998.

[37] P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information processing letters*, 6(3):80–82, 1977.

[38] P. van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Math. Systems Theory*, 10:99–127, 1977.

[39] D. Willard. Log-logarithmic worst-case range queries are possible in space $\Theta(N)$. *Information Processing Letters*, 17(2):81–84, 1983.

[40] D. E. Willard. New data structures for orthogonal range queries. *SIAM Journal on Computing*, 14:232–253, 1985.

[41] D. E. Willard. On the application of sheared retrieval to orthogonal range queries. In *Proceedings of the 2nd Annual ACM Symposium on Computational Geometry*, pages 80–89. ACM Press, 1986. See also technical report 86-4, SUNY albany.

[42] D. E. Willard. Applications of the fusion tree method to computational geometry and searching. In *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '92)*, pages 286–295. SIAM, 1992.

[43] D. E. Willard. Applications of range query theory to relational data base join and selection operations. *Journal of Computer and System Sciences*, 52, 1996.

[44] D. E. Willard. Examining computational geometry, Van Emde Boas trees, and hashing from the perspective of the fusion tree. *SIAM Journal on Computing*, 29(3):1030–1049, June 2000.

[45] D. E. Willard and G. S. Lueker. A data structure for dynamic range queries. *Information processing letters*, 15(5):209–213, 1982.

[46] D. E. Willard and G. S. Lueker. Adding range restriction capability to dynamic data structures. *Journal of the ACM*, 32(3):597–617, July 1985.

[47] A. C. Yao. Space-time tradeoff for answering range queries. In *ACM Symposium on Theory of Computing (STOC '82)*, pages 128–136, Baltimore, USA, 1982. ACM Press.

[48] A. C. Yao. On the complexity of maintaining partial sums. *SIAM Journal on Computing*, 14(2):277–288, 1985.