Space Efficient Dynamic Orthogonal Range Reporting

Yakov Nekrich*

Abstract

In this paper we present new space efficient dynamic data structures for orthogonal range reporting. The described data structures support planar range reporting queries in time $O(\log n + k \log\log(4n/(k+1)))$ and space $O(n \log\log n)$, or in time $O(\log n + k)$ and space $O(n \log^{\varepsilon} n)$ for any $\varepsilon > 0$. Both data structures can be constructed in $O(n \log n)$ time and support insert and delete operations in amortized time $O(\log^2 n)$ and $O(\log n \log\log n)$ respectively. These results match the corresponding upper space bounds of Chazelle [Ch88] for the static case.
We also present a dynamic data structure for d-dimensional range reporting with search time $O(\log^{d-1} n + k)$, update time $O(\log^d n)$, and space $O(n \log^{d-2+\varepsilon} n)$ for any $\varepsilon > 0$.

1 Introduction

Given a set of points $P \subset \mathbb{R}^d$, range reporting problem Q is to find all points in $P \cap Q$ for some set $Q \subset \mathbb{R}^d$. If set Q is a rectangle, the problem is called *orthogonal range reporting* problem. An example of orthogonal range reporting problem applied to databases is "Find all employees with income between 20000 and 30000 who are older than 40 and younger than 50 years old". See Willard [W96] for an extensive list of applications to database theory.

The time and space complexity of static and dynamic planar range reporting was considered in a number of papers. See [AE99] and [CT91] for surveys of work in this area. Chazelle [Ch88] has shown that this problem can be solved in time $O(\log n + k)$ and space $O(n \log^{\varepsilon} n)$ in the static case. Here and further k denotes the size of the answer. In [Ch88] two other space efficient solutions are also described. These solutions require $O(n \log\log n)$ space and $O(\log n + k \log\log(4n/(k+1)))$ time, or $O(n)$ space and $O(\log n + k \log^2(2n/(k+1)))$ time respectively. The above results are valid for the RAM model.

For the case of the pointer machine model (see e.g. [T79]), it was shown in [Ch90] that any range reporting data structure that works in $O(\log n + k)$ time

*Dept. of Computer Science, University of Bonn.Work partially supported by IST grant 14036 (RAND-APX). E-mail yasha@cs.uni-bonn.de

Source	Query Time	Space Usage	Insertion/ Deletion
Van Kreveld, Overmars, [vKO88]	$O(\sqrt{n \log n} + k)$	$O(n)$	$O(\log n)$
Van Kreveld, Overmars, [vKO89]	$O(\sqrt{n} \log n + k)$	$O(n)$	$O(\log n)$
Chazelle, [Ch88]	$O(\log n + k \log^2(2n/(k+1)))$	$O(n)$	–
Mehlhorn, [MN90]	$O(\log n \log\log n + k)$	$O(n \log n)$	$O(\log n \log\log n)$
Mortensen, [M03]	$O(\log n + k)$	$O(n \log n / \log\log n)$	$O(\log n)$
Chazelle, [Ch88]	$O(\log n + k \log\log(4n/(k+1)))$	$O(n \log\log n)$	–
Chazelle, [Ch88]	$O(\log n + k)$	$O(n \log^{\varepsilon} n)$	–
this paper	$O(\log n + k)$	$O(n \log^{\varepsilon} n)$	$O(\log^2 n)$/ $O(\log n \log\log n)$
this paper	$O(\log n + k \log\log n)$	$O(n \log\log n)$	$O(\log^2 n)$/ $O(\log n \log\log n)$

Table 1: Data Structures for Planar Orthogonal Range Reporting

requires at least space $\Omega(n \log n / \log\log n)$. An optimal upper bound for the pointer machine was also obtained (see [Ch86]).

Dynamic algorithms for geometric problems have gained increasing attention due to a number of important applications. However, the dynamic data structures are not so efficient as the static ones. An algorithm of Mehlhorn and Näher [MN90] answers planar range reporting queries in $O(\log n \log\log n + k)$ time and $O(n \log n)$ space, and supports update operations in $O(\log n \log\log n)$ time. Recently, Mortensen [M03] described a dynamic data structure with query time $O(\log n + k)$, update time $O(\log n)$, and space $O(n \log n / \log\log n)$.

In case of massive data sets it is sometimes important to answer orthogonal range queries using a linear (or almost linear) space data structure. If we accept polylogarithmic penalties for every answer, it is possible to answer range queries in $O((k+1)\log^2(2n/(k+1)))$ time and linear space [Ch88]; this data structure supports updates in $O(\log^2 n)$ time. Other linear space data structures do not have penalties for the answers at the cost of sufficiently higher query times. Van Kreveld and Overmars [vKO89] present a data structure with $O(\sqrt{n} \log n + k)$ query time. In [vKO88] they present a data structure with $O(\sqrt{n \log n} + k)$ query time and $O(\log n)$ update time.

More efficient algorithms exist for some special cases of planar range reporting. So, for instance, McCreight [McC85] describes an algorithm for three-sided range queries that requires linear space and $O(\log n + k)$ time.

In case of dynamic d-dimensional range reporting, $d \geq 3$, Willard [W87] has given an $O(\log^d n)$ query and update time, and $O(n(\log n / \log\log n)^d)$ space solution. [Ch88] describes the data structure with $O(\log^{d-1} n + k \log(2n/(k+1))^2)$ query time, $O(\log^d n)$ update time, and $O(n \log^{d-2} n)$ space . The technique of [MN90] extends to d dimensions, yielding a data structure with $O(\log^{d-1} n \log\log n + k)$ search and update times, and $O(n \log^{d-1} n)$ space.

1.1 Our Results

In this paper we present new space efficient dynamic data structures for orthogonal range queries. These data structures come very close to the linear space bound and at the same time support efficient range queries. In particular, we describe the first dynamic data structure with query time $O(\log n + k)$ that requires $O(n \log^{\varepsilon} n)$ space for any $\varepsilon > 0$. All previously described dynamic data structures that use $o(n \log n / \log\log n)$ space require either query time $\Omega(n^c)$ for $c > 0$ or penalties for every answer.

Theorem 1 *There exists a dynamic data structure supporting two-dimensional range reporting queries in time* $O(\log n + k)$ *and space* $O(n \log^{\varepsilon} n)$ *for any* $\varepsilon > 0$. *There exists a dynamic data structure supporting two-dimensional range reporting queries in time* $O(\log n + k \log\log \frac{4n}{k+1})$ *and space* $O(n \log\log n)$.

Both data structures can be constructed in $O(n \log n)$ *time and support insert and delete operations in amortized time* $O(\log^2 n)$ *and* $O(\log n \log\log n)$ *respectively.*

Results of Theorem 1 are valid[1] for the RAM model with word size logarithmic in n. Our results match the corresponding upper bounds of Chazelle [Ch88] for static data structures. Our data structures also improve on the result of Mehlhorn and Näher [MN90] both in terms of required space and of query time, and on the result of Mortensen [M03] in terms of space.

Many solutions of orthogonal range reporting problem are based on range trees and their variants, and fractional cascading. In this paper a different approach is used. As will be shown below, an orthogonal range query can be reduced to several three-sided range queries and/or an orthogonal range query on a smaller set of elements. The key to our space efficient solution is a novel dynamic range reduction technique, called *dynamic range reduction to extended rank space* . We believe that this technique is of interest on its own and can have other important applications.

We also extend our result for planar range searching to higher dimensions and obtain the following result:

Theorem 2 *For any* $\varepsilon > 0$ *and* $d \geq 3$, *there is a data structure* D' *that supports* d*-dimensional range reporting quer-ies in time* $O(\log^{d-1} n + k)$ *and requires* $O(n \log^{d-2+\varepsilon} n)$ *space;* D' *supports update operations in amortized time* $O(\log^d n)$ *and can be constructed in time* $O(n \log^{d-1} n)$.
For $d \geq 3$, *there is a data structure* D'' *that supports* d*-dimensional range reporting queries in time* $O(\log^{d-1} n + k \log\log \frac{4n}{k+1})$ *and requires* $O(n \log^{d-2} n \log\log n)$ *space;* D'' *supports update operations in amortized time* $O(\log^d n)$ *and can be constructed in time* $O(n \log^{d-1} n)$.

[1] We use $O(\log n + k \log\log \frac{4n}{k+1})$ instead of $O(\log n + k \log\log \frac{n}{k})$ in the second statement of the Theorem, to avoid $-\infty$, if $n = k$, or division by zero , if $k = 0$.

This result improves on the previously known results for dynamic d-dimensional range reporting.

The rest of this paper has the following structure. In section 2 we describe the data structure D with time and space bounds specified in Theorem 1. D requires existence of a dynamic data structure A which answers orthogonal range queries in logarithmic time and space $O(n \log^p n)$ for some constant $p > 0$. Constant p can be arbitrarily large. In section 3 a dynamic range reduction technique is presented. This technique allows us to reduce space requirements and, we believe, can have some other important applications. The analysis of the data structure D is finished in section 4. In section 5 we consider an extension of our results to d-dimensional queries, $d \geq 3$.

2 A data structure for orthogonal range reporting

In this section we give a rough description of our data structure.

Our solution is based on a recursive data structure representing rows and columns. A data structure containing m points is subdivided into $O(\sqrt{m/\log^p m})$ column data structures and $O(\sqrt{m/\log^p m})$ row data structures, each of which contains between $\sqrt{m \log^p m}/2$ and $2\sqrt{m \log^p m}$ points. Each point is stored in one row and one column data structure. For every row R_i and every column C_j we also store the list of elements from the set $K_{ij} = C_j \cap R_i$. For all rows R_i and columns C_j our data structure also supports three-sided range queries limited by one of vertical column borders or one of horizontal row borders . If the number of elements in a column or a row exceeds a certain $s_{min} = \Theta(\log n)$, we store the data structure D for elements from this column or row.

The data structure containing all elements will be further called a *level 0 data structure (D^0)*. Data structures corresponding to rows and columns of a level k data structure will be called level $k+1$ data structures. Observe that the number of elements in recursive data structure exponentially decreases with level l. Using the presented data structure we can either reduce an orthogonal query on $[1,..,n] \times [1,..,n]$ to a query in a single row (single column) or to a two-dimensional query in a data structure of size $O(n/\log^p n)$ and four three-sided queries on data structures of size $O(\sqrt{n \log^p n})$.

Below we give a more detailed description of the recursive data structure. Consider a data structure D^l on level l with m elements in the range $[1,..,O(m^3)] \times [1,..,O(m^3)]$. This range is subdivided into $\sqrt{m/\log^p m}$ columns $C_i = [c_{i-1}, c_i] \times [1,..,O(m^3)]$ and $\sqrt{m/\log^p m}$ rows $R_i = [1,..,O(m^3)] \times [r_{i-1},..,r_i]$ so that each column (row) stores between $\sqrt{m \log^p m}/2$ and $2\sqrt{m \log^p m}$ elements.

For every column and row we store a dynamic range reduction structure (explained in section 3) that maps point coordinates from range $[1,..,O(m^3)] \times [1,..,O(m^3)]$ to $[1,..,O(n^3)] \times [1,..,O(n^3)]$ where n is the maximum number of elements in a row (column). We also store level $l+1$ data structure for every column and row using coordinates in the reduced range.

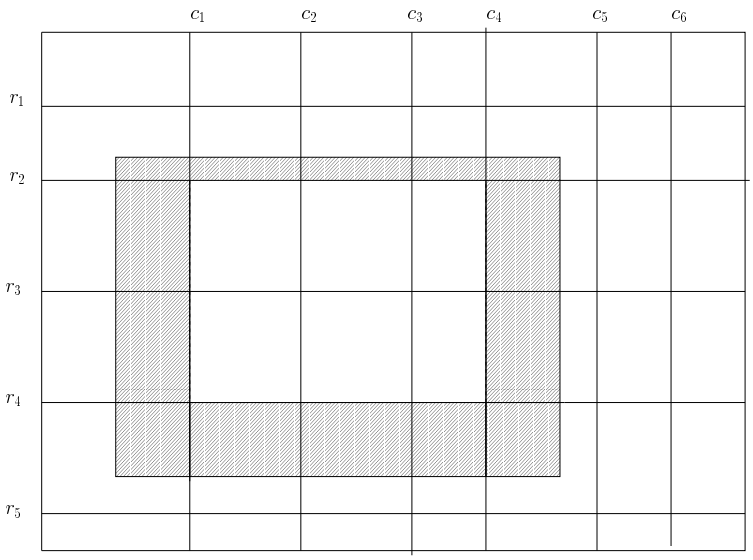

Figure 1: Range Query (Case 2). Areas of three-sided range queries are marked by slanted lines.

Every data structure D^l contains data structures for three-sided range queries for its columns and rows. Priority search trees of McCreight [McC85] are used for this purpose. D^l also contains data structure D_t with range $\sqrt{m/\log^p m} \times \sqrt{m/\log^p m}$ containing at most $m/\log^p m$ points. An element in D_t with coordinates i,j corresponds to a non-empty rectangle $K_{ij} = R_i \cap C_j$. We will use data structure A, which supports range queries in time $O(\log n)$, updates in $O(\log n)$ time, and requires $O(n \log^p n)$ space as D_t. For instance, the data structure from [M03] satisfies this condition and can be used as D_t. Since D_t contains at most $m/\log^p m$ elements it requires $O((m/\log^p m)\log^p m) = O(m)$ space.

We also need two one-dimensional structures D_r and D_c that store row borders r_i and column borders c_i respectively. Any linear space and logarithmic time data structure can be used for D_r and D_c.

A range query $[a,..,b] \times [c,..,d]$ to data structure D is processed as follows. First we use D_c and D_r to identify min_h and max_h (min_v and max_v) such that $c_{min_h-1} < a < c_{min_h} < c_{min_h+1} < \ldots < c_{max_h-1} < b < c_{max_h}$ ($r_{min_v-1} < c < r_{min_v} < r_{min_v+1} < \ldots < r_{max_v-1} < d < r_{max_v}$). We will call columns C_{min_h}, C_{max_h} and rows R_{min_v}, R_{max_v} marginal columns and marginal rows respectively. The rectangles K_{ij} with $min_h < i < max_h$ and $min_v < j < max_v$ will be called internal rectangles.

There are two possible cases

Case 1 $[a,..,b] \times [c,..,d]$ is contained in one row R_i or in one column C_j. That is, for some j, $c_j < a < b < c_{j+1}$, or for some i, $r_i < c < d < r_{i+1}$.

In this case we have to answer the same query for the data structure corresponding to C_j (R_i). Note that the number of elements has changed from m to $\sqrt{m \log^p m}$.

Case 2 $[a,..,b] \times [c,..,d]$ intersects with more than one row and more than one column. In this case we have to answer four three-sided queries for marginal rectangles. That is, we must report all elements p with $p \in C_{min_h} \cap [a,+\infty) \times [r_{min_v},..,r_{max_v-1}]$, or $p \in C_{max_h} \cap (-\infty,b] \times [r_{min_v},..,r_{max_v-1}]$, or $p \in R_{min_v} \cap [a,..,b] \times [c,+\infty)$ or $p \in R_{max_v} \cap [a,..,b] \times (-\infty,d]$. These three-sided queries can be answered using the corresponding priority search trees. We must also report all elements from non-empty internal rectangles. The non-empty rectangles can be found using the structure D_t.

In the first case, in time $O(\log n)$ the size of the data structure is reduced from n to $O(\sqrt{n \log n})$. In the second case, using structure D_t all non-empty rectangles K_{ij} can be found in time $O(\log n)$ and all points in the marginal rows and columns can also be found in $O(\log n)$ time. Therefore, if we ignore the time for reporting answers, we have the following recursive equation for the query time $T(n)$: $T(n) = min(2\log n + T(\sqrt{n \log^p n}), 2\log n + 4\log \sqrt{n \log^p n} + \log \sqrt{n/\log^p n})$ and $T(n) = O(\log n)$.

The space requirement of this data structure can be roughly estimated with $O(n \log^{O(1)} n)$. This follows from the fact that our data structure consists of $O(\log \log n)$ recursive levels and every point occurs in at most 2^l different data structures on recursive level l. However this rough analysis ignores the influence of the dynamic range reduction. In the following sections we will show how space requirements can be reduced. In section 3 a technique called *dynamic range reduction to extended rank space* will be described. With help of this technique a better analysis of space and time complexity will be performed in section 4.

3 Dynamic range reduction

The key idea of our space-reduction method is an order-preserving bijective dynamic mapping from the set S, such that $m/2 \leq |S| \leq 2m$, to $[1,..,O(m^3)]$. If we store instead of $x \in S$ its mapping $f(x)$, every element of S can be specified with $O(\log m^3)$ bits. Hence, any linear size data structure can be stored with $O(m \log m)$ bits. The technique for construction and maintenance of f is related to the solution of the list labelling problem ([IKR81]), which is also used in cache-oblivious B-trees ([BDF00]).

The mapping f is used as a base of our range-reduction technique, further called *dynamic range reduction to extended rank space*. This technique can be regarded as an extension of *reduction to rank space* technique (see, for instance, [Ch88]). First we describe the dynamic range reduction technique, the mapping itself will be described later in this section. Consider the set of planar points R and let R_x and R_y be the sets of x-coordinates and y-coordinates of all points

in R. Our two-dimensional range reduction structure consists of two dynamic mappings f_x and f_y for two coordinates. For each mapping we also store a linear space one-dimensional data structure that allows us to find predecessor and successor of an element in R_x or R_y in $O(\log n)$ time. We denote by $\hat{R}$ the set of points $\{(f_x(x), f_y(y))|(x,y) \in R\}$. Range query $Q = [x_1, .., x_2] \times [y_1, .., y_2]$ in R can be translated to a range query $\hat{Q} = [\hat{x}_1, .., \hat{x}_2] \times [\hat{y}_1, .., \hat{y}_2]$ in $\hat{R}$, where $\hat{x}_1 = f_x(x'_1) + c_f/2$, $\hat{y}_1 = f_y(y'_1) + c_f/2$, $\hat{x}_2 = f_x(x'_2) - c_f/2$, $\hat{y}_2 = f_y(y'_2) - c_f/2$ and x'_1, y'_1 and x'_2, y'_2 are predecessors and successors of x_1, y_1 and x_2, y_2 respectively. Constant c_f is such that $|f(x) - f(y)|geqc_f$, for all x, y. Obviously, $(x,y) \in Q \cap R \Leftrightarrow (f_x(x), f_y(y)) \in \hat{Q} \cap \hat{R}$.

We also associate a time-stamp t_i with every element $\hat{e}_i \in \hat{R}$. When R with $|R| = m$ is constructed its elements $e_1, e_2, \ldots, e_m$ get time-stamps $1, 2, \ldots, m$. When a new element is added to R, the corresponding element in $\hat{R}$ gets time-stamp $t_{max} + 1$, where t_{max} is the maximum time-stamp previously used. With every element $\hat{e}_i$ in $\hat{R}$ we store its time-stamp t_i. We also store an array Inv with at most $2m$ entries, so that $Inv[i] = e_i = (x_i, y_i)$ such that $\hat{e}_i = (f_x(x_i), f_y(y_i))$ has time-stamp i. This allows us to find for every $(x,y) \in \hat{R}$ $f^{-1}(x,y)$ in constant time.

Now we turn to the description of mapping $f : S \to [1, .., v_{max}]$, for a dynamic set S such that $m/2 \leq |S| \leq 2m$. Here $v_{max} = 4c_f m^3$ for some constant c_f. We presume that every element $x \in S$ has an associated integer value $val(x)$ so that $\forall x, y \in S$ $val(x) < val(y) \Rightarrow f(x) < f(y)$, but if $val(x) = val(y)$ $f(x) \neq f(y)$. Elements can be added to or deleted from set S, and when S is updated values $f(x)$ for some $x \in S$ may change. We say that element x is *moved* if the value $f(x)$ has changed. We will show that f can be constructed in $O(|S|)$ time if S is sorted. We also show that update operations can be performed in amortized time $O(\log^2 m / \log\log m)$. That is, we will show that when an element is inserted into S or deleted from S, we can change the values $f(x)$ of $O(\log^2 m / \log\log m)$ elements x so that the properties of mapping f are preserved.

Lemma 1 *For a set S of size m, mapping f can be constructed in $O(m)$ time, if S is sorted. If $m/2 \leq |S| \leq 2m$, insertions of new elements can be performed in amortized time $O(\log^2 m / \log\log m)$. Deletions can be performed in amortized time $O(1)$.*

Proof: We can set $f(x) = i \cdot m^2 \cdot c_f$, if x is the i-th element in ascending order. This proves the first par of Lemma 1.

For ease of description we say that x belongs to interval $I \subset [1, .., v_{max}]$ if $f(x) \in I$. Distance between two neighbor elements x and y $(d(x,y))$ will denote the difference $|f(y) - f(x)|$. For an interval I the *potential function* $pot(I)$ equals to $\sum_{x \in I} \log d(x, succ(x))/c_f$, where $succ(x)$ denotes the successor of x. *Average potential* $Av(I)$ is the logarithm of the average distance between neighbor elements in I:

$$Av(I) = \log(\sum_{x \in I} d(x, succ(x))/c_f |\{x|f(x) \in I\}|)$$

.

First we show how insertions are processed. Let $I_j^k = [(j-1)m^2 c_f \log^k m, .., jm^2 c_f \log^k m]$. We say that interval I is *rebuilt*, if all elements of I are moved so that for all elements $x, y \in I$ $d(x, succ(x)) = d(y, succ(y))$. In other words, when interval I is rebuilt, the average potential of I becomes "uniform" on this interval, or $\forall\ I_j^0, I_{j'}^0 \subset I:\ Pot(I_j^0) = Pot(I_{j'}^0)$.

Suppose that element z must be inserted, and let x and y be its predecessor and successor respectively. If $d(x, y) \geq 2c_f$, we set $f(z) = f(x) + d(x, y)/2$. Otherwise we rebuild the interval I_m^0. We also keep track of the number of elements inserted into intervals I_j^k for $k \geq 1$. If more than $\log^{k+1} m$ elements were inserted into I_j^k since I_j^k or some $I_{j'}^{k'} \supset I_j^k$ was rebuilt for the last time, we rebuild I_j^k and split it into $\log m$ intervals containing at most $\log^k m$ elements.

It is easy to see that the average potential of every interval I_j^0 after a rebuild always exceeds $\log m$. An insertion decreases the average potential of the corresponding interval I_j^0 by at most 1. Therefore every I_j^0 will be rebuilt at most $\log m$ times between two rebuilds of $I_{j'}^1 \supset I_j^0$. The total number of elements in I_j^0 does not exceed $\log^2 m + \log m$. Rebuilding interval I takes time $|\{x | x \in I\}|$. Hence after a sequence of m insertions, intervals I_j^0 will be rebuilt at most $m/\log m$ times, and every rebuilding takes $O(\log^2 m)$ time. Intervals I_j^k are rebuilt after $\log^{k+1} m$ insertions, and the cost of rebuilding is $O(\log^{k+2} m)$.

Therefore after m elements were inserted, total time for rebuilding all intervals does not exceed $O(\log^2 m)\frac{m}{\log m} + O(\log^3 m)\frac{m}{\log^2 m} + \ldots + O(\log^{s+1} m)\frac{m}{\log^s m}$, where $s = \lceil \log_{\log m} m \rceil$. Hence, total time for rebuilds is $O(m \log^2 m / \log\log m)$. The details of the rebuilding procedure will be given in the full version of this paper.

When an element z is deleted we simply "free" the value of $f(z)$, i.e., we set $d(x, y) = d(x, z) + d(z, y)$, where x is the predecessor of z and y is the successor of z.

□

After a sequence of m delete and insert operations, mapping f can be rebuilt in $O(m)$ time according to the first part of Lemma 1. This incurs additional amortized cost $O(1)$.

4 Analysis

The space requirements of data structure D can be sufficiently reduced by using the reduction to extended rank space. With help of this technique, elements of the data structure D^l can be stored in $O(\log(D^l)$ bits.

Lemma 2 *Data structure D with n elements can be constructed in $O(n \log n)$ time.*

Proof: Let $g(n) = \sqrt{n \log^p n}$. If sorted lists of points for D are available sorted lists of elements for columns C_j and rows R_i can be constructed in $O(n)$

time. $4n/g(n)$ priority search trees for rows and columns can also be constructed in $O(n)$ time as well as the structures for dynamic range reduction. The time required to construct priority search trees for D_c and D_r is $O(n^{1/2}\log n^{1/2}) = o(n)$. D_t can be constructed in $O(n)$ time.

We get the following recurrent relation for construction time $K(n)$: $K(n) = O(n) + 2(n/g(n))K(g(n)) = O(n\log n)$. Hence $K(n) = O(n\log n)$. □

Proofs of Lemmas 3 and 5 are provided in Appendix A.

Lemma 3 *Data structure D requires $O(n\log\log n)$ space*

Lemma 4 *Using D, orthogonal range queries can be answered in $O(\log n + k\log\log n)$ time.*

Proof: Consider a query $Q = [a,..,b]\times[c,..,d]$. The search in D^0 begins by looking for the columns and rows that intersect with $[a,..,b]\times[c,..,d]$. If the whole rectangle $[a,..,b]\times[c,..,d]$ is contained in one row or in one column (Case 1 described in section 2) the search is continued in the data structure on the next level. As follows from the discussion in Lemma 3, after at most $\log\log n + c_l$ recursion levels the search is either reduced to search in a data structure of size $O(\log n)$, or the situation corresponding to Case 2 of section 2 is achieved. In the first case query can be answered in time $O(\log n)$ by exhaustive search. Recall that in Case 2 of section 2 query rectangle intersects with more than one row and more than one column, and the query can be answered by answering four three-sided queries and one range query to D_t. Hence in $O(\log n)$ time the answers can be found. However, we have only found the point coordinates on some level l. To find the original coordinates, the reverse range reduction must be applied l times. Since there are at most $O(\log\log n)$ recursive levels, and range reduction can be reversed in a constant time, the costs for finding the correct coordinates of every answer are at most $O(\log\log n)$. □

Lemma 5 *Insert operations into data structure D can be performed in $O(\log^2 n)$ amortized time. Deletions from D can be performed in $O(\log n\log\log n)$ amortized time.*

A combination of Lemmas 3, 4, and 5 almost proves the second part of Theorem 1. We can reduce query time from $O(\log n + k\log\log n)$ to $O(\log n + k\log\log\frac{4n}{k+1})$ applying the technique described in [Ch88], section 5.

We also describe a modification of the above data structure, which requires slightly more space than the previous variant, but allows us to avoid the $O(\log\log n)$ penalty for every answer. In modified data structure D' we apply range reduction only on every r-th level for a parameter r. If we choose $r = \varepsilon\log\log n$, for a constant ε, then range reduction is applied only a constant number of times. Hence, queries can be answered in time $O(\log n + k)$.

Lemma 6 *Data structure D' requires space $O(n\log^{\varepsilon} n)$.*

Lemma 7 *Insert operations into data structure D' can be performed in $O(\log^2 n)$ amortized time. Deletions from D' can be performed in $O(\log n\log\log n)$ amortized time.*

Two above lemmas are also proven in Appendix B. Combination of these two lemmas proves the first part of Theorem 1.

5 Range Searching in Higher Dimensions

We use a standard technique for extending d-dimensional queries to $(d+1)$-dimensional queries (cf. e.g., [Ch88]).

Let $P \subset \mathbb{R}^{d+1}$. Let P_{d+1} be the set of the $d+1$-st coordinates of points in P. For convenience we assume that $a_{d+1} \neq b_{d+1}$ for all $a, b \in P$. We construct a balanced binary tree T whose leaves are associated with elements of P_{d+1}, so that leaves are ordered left to right. We associate with each internal node N a *range* $[a_N, .., b_N]$ such that all leaf descendants of N belong to $[a_N, .., b_N]$. In every node N we also store a d-dimensional data structure D_N. D_N contains points $e \in P$, $v_i < e_{d+1} \leq v_j$, but only first d coordinates of elements are stored. An arbitrary interval $[a, .., b]$ can be represented as a union of $O(\log n)$ intervals $[a_{N_i}, .., b_{N_i}]$, where $[a_{N_i}, .., b_{N_i}]$ is an interval associated to node N_i. Therefore a $(d+1)$-dimensional query $[a, .., b] \times Q$, where Q is a d-dimensional range, can be reduced to $O(\log n)$ d-dimensional queries to data structures corresponding to nodes N_i. Thus query time is $O(q(n) \log n + kp(n)$, if a d-dimensional query time is $O(q(n)+kp(n))$. Since every element is stored in $\lceil \log n \rceil$ data structures, space requirement increases by a logarithmic factor, compared to a d-dimensional data structure. It can be shown (see, e.g., [W85]) that the (amortized) update time also increases by an $O(\log n)$ factor.

Applying the above technique $d-2$ times to the results of Theorem 1, we immediately obtain Theorem 2.

6 Conclusion

In this paper space- and time-efficient algorithms for orthogonal range reporting were presented. We believe that the dynamic range reduction technique presented here will also find applications in other geometric data structures.

We also suppose that using deamortization techniques amortized time bounds for our data structures can be turned into the corresponding worst case time bounds.

The presented data structure has $O(\log^2 n)$ insertion time, due to the $O(\log^2 n)$ insertion time of the dynamic range reduction component. If we will be able to improve the performance of the dynamic range reduction technique, this would lead to better insertion times in our data structure.

Acknowledgments

The author thanks Marek Karpinski for helpful remarks and discussions.

References

[AE99] P. K. Agarwal and J. Erickson "Geometric range searching and its relatives" In B. Chazelle, J. E. Goodman, and R. Pollack, editors, "Advances in Discrete and Computational Geometry", vol. 23 of Contemporary Mathematics, 1–56. AMS Press, Providence, RI, 1999. Available at `http://citeseer.ist.psu.edu/article/agarwal99geometric.html`.

[BDF00] M. A. Bender, E. D. Demaine, M. Farach-Colton "Cache-Oblivious B-Trees", Proc. 41st FOCS, 399-409 , 2000.

[Ch86] B. Chazelle "Filtering Search: A New Approach To Query Answering", SIAM J. on Computing, 1986, vol. 15, 703-724.

[Ch88] B. Chazelle "A Functional Approach to Dynamic Data Structures", SIAM J. on Computing, vol. 17, 1988, 427-462.

[Ch90] B. Chazelle "Lower Bounds for Orthogonal Range Search II. The Arithmetic Model", J. of the ACM, vol. 37, 1990, 439 - 463.

[CT91] J. L. Chiang, R. Tamassia " Dynamic Algorithms in Computational Geometry", Technical Report CS-91-24, Dept. of Computer Science, Brown University, 1991.

[IKR81] A. Itai, A. G. Konheim, M. Rodeh "A Sparse Table Implementation of Priority Queues", Proc. 8th ICALP,417-431 ,1981.

[vKO88] M. Van Kreveld and M.H. Overmars " Divided K-d Trees", Tech. Report RUU-CS-88-28, Dept. of Comp. Science, Univ. of Utrecht.

[vKO89] M. Van Kreveld and M.H. Overmars "Concatenable structures for Decomposable problems" Tech. Report RUU-CS-89-16, Dept. of Comp. Science, Univ. of Utrecht.

[L78] G. S. Lueker " A Data Structure for Orthogonal Range Queries", 19th ACM Symp. on Foundations of Computer Science, 28-34, 1978.

[M84] Kurt Mehlhorn "Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry" Springer-Verlag New York, Inc., New York, NY, 1984.

[MN90] K. Mehlhorn and S. Näher "Dynamic Fractional Cascading" , Algorithmica vol. 5, 1990, 215-241.

[McC85] E.M. McCreight "Priority Search Trees", SIAM J. on Computing, vol. 14, 1985, 257-276.

[M03] C. W. Mortenesen "Fully Dynamic Two Dimensional Orthogonal Range and Line Segment Intersection Reporting in Logarithmic Time" Proc. of the 14th ACM-SIAM Symposium on Discrete Algorithms, 618-627, 2003.

[O87] M. H. Overmars "Design of Dynamic Data Structures" Springer-Verlag New York, Inc., Secaucus, NJ, 1987.

[T79] R.E. Tarjan "A Class of Algorithms Which Require Nonlinear Time to Maintain Disjoint Sets", JCSS, vol. 18, 1979, 110-127.

[W85] D. E. Willard "New Data Structures for Orthogonal Range Queries", SIAM J. on Computing, vol. 14, 1985, 232-253.

[W87] D. E. Willard "Multidimensional Search Trees That Provide New Types of Memory Reductions", J. of the ACM, vol. 34, 1987, 846-858

[W96] D. E. Willard "Applications of Range Query Theory to Relational Data Base Join and Select Operations" Journal of Computer and System Sciences, vol. 52, 1996, 157-169.

A Proofs of lemmata in section 4

Proposition 1 *Let $s^k(n)$ be the maximum number of elements in a data structure on level k. Let l_{max} be the maximum number of levels in D. Then*
$s^k(n) = O(n^{1/2^k} \log^p n \sqrt{\log\log n})$
$\log s^k(n) = O(1/2^k \log n + (p+1)\log\log n)$
$\sum_{l=0}^{l_{max}} 2^l \log^2(s^l(n))/\log\log s^l(n) = O(\log^2 n)$

Proof: The data structure on level 0 requires $n \log n$ bits, since number of stored values is n and each stored value requires $\log n$ bits. Now suppose that data structures on level k contain $s^k(n) \leq cn^{1/2^k} \log^p n \sqrt{\log\log n}$ elements for some constant c. In this case $\log s^k(n) < (1/2^k \log n + (p+1)\log\log n)$. Then:

$$s^{k+1}(n) = (s^k(n)\log(s^k(n)))^{1/2} <$$

$$(cn^{1/2^k} \log^p n \log\log n(1/2^k \log n + (p+1)\log\log n))^{1/2} <$$

$$\sqrt{c} n^{1/2^{k+1}} \sqrt{\log^p n \log\log n}\sqrt{\log n} <$$

$$\sqrt{c} n^{1/2^{k+1}} \log^p n \sqrt{\log\log n} <$$

$$cn^{1/2^{k+1}} \log^p n \sqrt{\log\log n}$$

and

$$\log s^{k+1}(n) < (1/2^{k+1}\log n + (p+1)\log\log n)$$

This proves the first two equalities.

It immediately follows from the first two inequalities that $\sum_{l=0}^{l_{max}} 2^l \log(s^l(n)) = O(\log\log n \log n)$. On the other hand, $s^k(n) > n^{1/2^k}$ and $\log\log s^k(n) > (\log\log n - k)$. Also recall that $s^k(n) > \log n$ for $k < l_{max}$ by definition of l_{max}, hence $s^k(n) > \log\log\log n$. $\sum_{l=0}^{l_{max}} \log s^l(n)/\log\log s^l(n) =$

$\sum_{l=0}^{l_{mid}} \log s^l(n)/\log\log s^l(n) + \sum_{l=l_{mid}+1}^{l_{max}} \log s^l(n)/\log\log s^l(n)$, where $l_{mid} = \log\log n/2$. Now we estimate the first and the second summands separately.

$$\sum_{l=0}^{l_{mid}} \log s^l(n)/\log\log s^l(n) <$$

$$\sum_{l=0}^{l_{mid}} 2(\log n/2^l + (p+1)\log\log n)/c'\log\log n =$$

$$O(\log n/\log\log n)\sum_{l=0}^{l_{mid}} 1/2^l = O(\log n/\log\log n)$$

And the second summand is:

$$\sum_{l=l_{mid}+1}^{l_{max}} \log s^l(n)/\log\log s^l(n) <$$

$$\sum_{l=l_{mid}+1}^{l_{max}} (\log n/2^{\log\log n/2})/\log\log s^l(n) <$$

$$< (\log\log n/\log\log\log n)\sqrt{\log n} =$$

$$O(\log n/\log\log n)$$

. Now we can express

$$\sum_{l=0}^{l_{max}} 2^l \log^2(s^l(n))/\log\log s^l(n) =$$

$$\sum_{l=0}^{l_{max}} (2^l \log(s^l(n))) \times (\log(s^l(n))/\log\log s^l(n))$$

$$< \sum_{l=0}^{l_{max}} 2^l \log(s^l(n)) \sum_{l=0}^{l_{max}} \log(s^l(n))/\log\log s^l(n)$$

The last expression is $O(\log n \log\log n)O(\log n/\log\log n) = O(\log^2 n)$. □

Lemma 3 *Data structure D requires $O(n\log\log n)$ space.*

Proof: Using dynamic range reduction to extended rank space, we can express elements of D with $O(\log|D|)$ bits. Then it follows from the Proposition 1 that every data structure on level l contains no more than
$O(n^{1/2^l}\log^p n\sqrt{\log\log n})$ elements and every element can be stored with
$(1/2^l)\log n + (p+1)\log\log n$ bits.

The total number of bits required by all data structures on level l can be above bounded by $c'(n\log n + n2^l(p+1)\log\log n)$, where c' is a constant. Besides

that, $s^l(n) \leq \log n$ for all $l > l_{max}$. Here $l_{max} = \log\log n + c_l$ for some constant c_l and l_{max} is the number of levels in the data structure. The total number of bits is $\sum_{l=0}^{l_{max}} c'(n \log n + n2^l(p+1)\log\log n) = O(l_{max} n \log n + \sum 2^l n \log\log n)$ bits. Since the number of recursive levels $l_{max} = \log\log n + c_l$, the total number of bits is $O(n \log n \log\log n)$. Thus we need $O(n \log\log n)$ words of size $\log n$.

□

Lemma 5 *Insert operations into data structure D can be performed in $O(\log^2 n)$ amortized time. Deletions from D can be performed in $O(\log n \log\log n)$ amortized time.*

Proof: Suppose that a new element e is inserted. Then it is inserted into $O(\log n)$ range reduction structures, $O(\log n)$ priority search trees and $O(\log n)$ structures D_t. As explained above a new point must be inserted into one data structure on level 0, two data structures on level 1 and so on. Hence e must be inserted into 2^l range reduction structures, $O(2^l)$ priority trees and 2^l structures D_t on level l. Since insertion into a range reduction structure is the most time-consuming operation, it is enough to estimate the total time for modifying all dynamic range reduction structures. Every insertion of an element to a range reduction structure of size m can be performed by modifying values of $O(\log^2 m / \log\log m)$ other elements (on average). These modifications can be implemented in $O(log^2 m / \log\log m)$ time.

Thus the total number of operations can be limited by
$\sum_{l=0}^{l_{max}} 2^l \log^2(s^l(n)) / \log\log s^l(n)$ According to Proposition 1 the last expression can be limited by $O(\log^2 n)$.

The cost of a deletion can be estimated in a similar way. Indeed, if a point is deleted it must be deleted from $O(2^l)$ priority trees on level l, from 2^l range reduction structures and from at most 2^l structures D_t Since deleting of an element from a range reduction structure leads on average to $O(1)$ deletions from priority trees the total time is $O(\sum_{l=0}^{l_{max}} 2^l \log(s^l(n))) = O(\log n \log\log n)$.

In the above analysis we ignored the costs of rebuilding D (global rebuilding) or parts of D (local rebuilding). Suppose the number of elements after the last global rebuilding was n_0. Then the next global rebuilding takes place when $n_0/2 > |D|$ or $3n_0/2 < |D|$. During the global rebuilding the whole data structure is reconstructed "from scratch". As was shown in Lemma 2 this incurs total cost of $O(n \log n)$ and amortized cost $O(\log n)$.

Consider some D^l on level l containing m elements. Every row and column of D^l can hold between $\sqrt{m \log^p m}/2$ and $2\sqrt{m \log^p m}$ elements. If after a series of updates the number of elements in a column C_i violates these bounds, we consider one of the neighbor columns C_{i+1} and C_{i-1}. Since $\sqrt{m \log^p m} \leq |C_i \cup C_{i+1}| < 4\sqrt{m \log^p m}$ (the same bounds are also true for C_{i-1}) we can construct v columns from elements of C_i and C_{i+1} for $1 \leq v \leq 4$. Since these columns can be rebuilt in time $O(\sqrt{m \log^p m} \log \sqrt{m \log^p m})$ (see Lemma 2) rebuilding columns incurs amortized cost $O(\log m)$. Rebuilding rows is, of course, identical to rebuilding columns.

When a row or a column is rebuilt, D_t must also be updated. Namely, up to $O(\sqrt{m/\log^p m})$ elements are inserted to or deleted from D_t. The total cost of updating D_t is $O(\sqrt{m/\log^p m}\log m) = O(\sqrt{m\log^p m})$. Thus rebuilding D_t incurs only constant amortized cost and the total amortized cost of insertion or deletion is $O(\log m)$. Now recall that when an element e is updated it is updated in 2^l structures D^l on level l. Applying the same analysis as above we see that the total amortized rebuilding cost is $\sum 2^l O(\log(s^l(n))) = O(\log n \log\log n)$.

Therefore taking into consideration amortized costs for local and global rebuilding does not change the time bounds of update operations. □

Lemma 6 *Data structure D' requires space $O(n\log^\varepsilon n)$.*

Proof: Recall that in D' we apply range reduction only on every r-th level for some parameter r. In this case the total number of bits used by level $l = ir + q$ data structures is $cn2^l \log s^{ir}(n)$ for some constant c. Then the total number of bits on levels $ir, ir+1, \ldots, (i+1)r-1$ is
$\sum_{q=0}^{r-1} cn2^{ir+q}\log s^{ir}(n) cn2^{ir}\log s^{ir}(n)\sum 2^q$. Since $\log(s^{ir}) < c'(1/2^{ir}\log n + (p+1)\log\log n)$ the total number of bits on levels $ir, ir+1, \ldots, (i+1)r-1$ is $O(n2^r\log n + 2^{(i+1)r}\log\log n)$. Then the total number of bits on all levels equals to $O((l_{max}/r)n2^r\log n + n\sum_{i=1}^{l_{max}/r} 2^{(i+1)r}\log\log n)$. The second summand is $O(n\log n\log\log n)$. Since $l_{max} \le \log\log n + c_l$, we can choose $r = \varepsilon\log\log n$ and the first summand is $O((1/\varepsilon)\log^{1+\varepsilon} n)$. Thus the total number of bits does not exceed $O(n\log n\log^\varepsilon n)$ and the modified data structure can be stored in $O(n\log^\varepsilon n)$ words of memory. □

Lemma 7 *Insert operations into data structure D' can be performed in $O(\log^2 n)$ amortized time. Deletions from D' can be performed in $O(\log n\log\log n)$ amortized time.*

Proof is analogous to the proof of Lemma 5.