Dynamic Planar Orthogonal Range Reporting

Marek Karpinski* Yakov Nekrich†

Abstract

In this paper we present a dynamic data structure for planar orthogonal range reporting with query time $O(\log n/\log\log n + k)$ and space $O(n\log^{\varepsilon} n)$ for any $\varepsilon > 0$ and k the answer size. We also present a space efficient dynamic data structure with $O(\log n/\log\log n + k\log\log n)$ query time that uses $O(n\log\log n)$ space. These are the first dynamic data structures with sublogarithmic query time for that problem.

Keywords: Algorithms and Data Structures, Computational Geometry, Dynamic Data Structures, Range Reporting

1 Introduction

The planar orthogonal range reporting problem is to maintain a set of points S so that for an arbitrary query rectangle Q all points from S that belong to Q can be reported. This problem has been studied extensively; surveys of the previous results are given in [2] and [12].

Several space efficient static data structures for this problem with logarithmic query time are described in Chazelle [10]. In particular, [10] describes a data structure with $O(n\log^{\varepsilon} n)$ space and $O(\log n + k)$ time and a data structure with $O(\log n + k\log\log\frac{4n}{k+1})$ query time and $O(n\log\log n)$ space; here and further $k = |S \cap Q|$ is the size of the answer. Using a dynamization of the fractional cascading technique of Chazelle and Guibas [9], Mehlhorn and Näher [19] described a dynamic data structure with query time $O(\log n\log\log n + k)$, update time $O(\log n\log\log n)$ and space $O(n\log n)$. Mortensen [21] described a data structure that requires $O(n\log n/\log\log n)$ space and supports queries and updates in $O(\log n + k)$ and $O(\log n)$ time respectively. In [22] the space requirements for the

*Dept. of Computer Science, University of Bonn. E-mail marek@cs.uni-bonn.de. Work partially supported by a DFG grant, Max-Planck Research Prize, and IST grant 14036 (RAND-APX).

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

dynamic case are further reduced: the data structures of [22] use either $O(n \log^{\varepsilon} n)$ space and support queries in $O(\log n + k)$ time, or $O(n \log\log n)$ space and $O(\log n + k \log\log n)$ query time, thus matching the two above mentioned results of Chazelle [10] for the static case.

In the case of orthogonal range reporting on an $n \times n$ grid, there exist static data structures with sublogarithmic query time. For instance, Overmars [24] described a data structure with query time $O(\log\log n + k)$ using space $O(n \log n)$; in [3] a data structure with query time $O(\log\log n + k)$ and space $O(n \log^{\varepsilon} n)$ is described. Using the reduction to rank space technique(see, e.g., [10]) and data structures for predecessor queries, a sublogarithmic time can be achieved for the general case static data structures. Combining the fusion trees of Willard with the result of [3], we obtain a $O(\log n / \log\log n + k)$ query time and $O(n \log^{\varepsilon} n)$ space data structure; another $O(\log n / \log\log n + k)$ time data structure was presented by Willard [29]. Combining the exponential search trees (see [4], [5],[6]) with [3], we obtain a data structure with $O(\sqrt{\log n / \log\log n} + k)$ query time.

In the case of dynamic planar orthogonal range reporting queries, the fastest previously known dynamic data structures (cf. [21], [22]) have query time $\Omega(\log n + k)$. In this paper we present a dynamic data structure with $O(\log n / \log\log n + k)$ query time for planar range reporting queries. To the best of our knowledge, this is the first data structure with $o(\log n + k)$ query time.

Our approach depends on a reduction of a planar range reporting query to several three-sided queries[1] and a planar range reporting query on a set with a much smaller number of elements.

1.1 Our Results

We start with formulating our main results.

Theorem 1 *Let $a = \log_{3/2} 3 \approx 2.71$. There is a data structure D that supports planar orthogonal range reporting queries in $O(\log n / \log\log n + k)$ time and updates in $O(\log^a n)$ time, where k is the size of the answer. D can be constructed in $O(n \log n)$ time and requires $O(n \log^{\varepsilon} n)$ space for arbitrary $\varepsilon > 0$.*

Theorem 2 *Let $a = \log_{3/2} 3 \approx 2.71$. There is a data structure D' that supports planar orthogonal range reporting queries in $O(\log n / \log\log n + k \log\log n)$ time and updates in $O(\log^a n)$ time, where k is the size of the answer. D' can be constructed in $O(n \log n)$ time and requires $O(n \log\log n)$ space.*

Our results are valid in the unit cost RAM; we assume that all point coordinates are integers, but the size of point coordinates is not limited.

[1] A three-sided query will be defined in the section 1.1

The result of Theorem 1 is a $\log\log n$ factor improvement in query time compared to [22] and an improvement in terms of both space and query time compared to [21].

A *three-sided* range reporting query is a special case of the planar range reporting query, in which one side of the query rectangle is constrained to lie on one of the axes. Our result is based on two general reductions. The first reduction converts a dynamic data structure for three-sided queries into a dynamic data structure for planar range reporting. The second reduction converts a dynamic data structure with space $\Theta(n \log^p n)$ into a dynamic data structure with space $O(n \log^\varepsilon n)$ for arbitrary constants $p > 0$, $\varepsilon > 0$.

- Suppose there is a data structure B for three-sided range queries with query time $O(t(n)+k)$, where $t(n) = \Omega(\sqrt{\log n/ \log\log n})$, and update time $u(n)$ that uses space $O(n)$ and can be constructed in linear time. Then there exists a data structure A for planar range reporting with query time $O(t(n) + k)$ and update time $u'(n)$, such that $u'(n) = O(u(n)) + 3u'(n^{2/3})$. A can be constructed in $O(n \log^2 n)$ time and uses $O(n \log^2 n)$ words of memory.

- Suppose there exist a dynamic data structure A for planar range reporting queries with $O(t(n)+k)$ query time, $t(n) = \Omega(\sqrt{\log n/ \log\log n})$ and $O(n \log^p n)$ space, and a dynamic linear space data structure B for three-sided queries with query time $O(t(n) + k)$. Then there exists a dynamic data structure D for planar range reporting queries with query time $O(t(n) + k)$, and space $O(n \log^\varepsilon n)$.

The condition $t(n) = \Omega(\sqrt{\log n/ \log\log n})$ means that the query time cannot be asymptotically faster than the time to answer a predecessor query (see [6]). Basically, we show that any dynamic linear-space data structure for three-sided queries whose query time is not asymptotically faster than the lower bound on predecessor queries can be converted into a dynamic data structure for planar orthogonal range reporting queries with the same query time and only a small increase in space. The precise description of reductions is given in Theorems 4 and 5. These reductions are of interest on their own, since they elucidate the connection between data structures for three-sided range queries and (space efficient) data structures for general planar range reporting queries.

2 An $O(\log n/ \log\log n + k)$ Time Data Structure

In this section we describe a data structure that achieves query time $O(\log n/ \log\log n + k)$ but requires $O(n \log^2 n)$ space.

We will use the following lemma from [29]:

Lemma 1 *There is a dynamic linear space data structure that supports three-sided queries in $O(\log n/\log\log n+k)$ time, updates in $O(\log n/\log\log n)$ amortized time, and can be constructed in $O(n)$ time.*

Following [29], we call the data structure from Lemma 1 *a fusion priority tree.*

Theorem 3 *There exists a dynamic data structure A that supports planar orthogonal range reporting queries in time $O(\log n/\log\log n + k)$, update operations in $O(\log^a n)$ time for $a = \log_{3/2} 3 \approx 2.71$. A uses $O(n\log^2 n)$ words of memory and can be constructed in $O(n\log^2 n)$ time.*

In the data structure described in this section the set of points stored in A is divided into columns and rows. The set of points is divided into columns $C_i = [c_{i-1}, c_i] \times (-\infty, +\infty)$ so that C_i contains between $n^{2/3}/2$ and $2n^{2/3}$ points; the number of columns is $O(n^{1/3})$. In the same way the set of points is divided into $O(n^{2/3})$ rows $R_i = (-\infty, +\infty) \times [r_{i-1}, r_i]$, so that each row contains between $n^{2/3}/2$ and $2n^{2/3}$ elements.

Our data structure consists of the following components:

1. For every column and every row we store two fusion priority trees of Willard [29] that allow us to answer three-sided range queries in $O(\log n/\log\log n)$ time. For every row R_i, data structures that answer queries $(r_{i-1}, d] \times [a, b]$ and $[d, r_i) \times [a, b]$ are stored. For every column C_j, data structures that answer queries $(c_{j-1}, b] \times [c, d]$ and $[b, c_j) \times [c, d]$ are stored.

2. We store data structures T_c and T_r for one-dimensional predecessor queries with $O(\log n/\log\log n)$ query time. T_c and T_r store all column borders c_i and all row borders r_i respectively. T_c and T_r can be implemented using e.g. the fusion tree of Willard.

3. Let $K_{ij} = C_j \cap R_i$. We also store data structure A_t that stores all points (i, j) such that $K_{ij} \neq \emptyset$. Data structure A_t is defined recursively, i.e., A_t consists of the same type of components as our initial data structure$A$. Observe that A_t contains at most $O(n^{2/3})$ elements due to the fact that the number of non-empty K_{ij} is at most $n^{1/3}n^{1/3}$.

4. For every row R_i and every column C_j, a recursively defined data structure $R_i(A)$ $(C_j(A))$ is stored. $R_i(A)$ $(C_j(A))$ contains all points from row R_i (column C_j) of A. A top level data structure is called the *level 0* data structure. A data structure that stores elements from a column or a row of some level l data structure A^l, or a data structure $(A^l)_t$ are called level $l + 1$ data structures.

5. Consider a level l data structure A^l. A point p in a level $l+1$ data structure $(A^l)_t$ *contains* all points in K_{ij}. A point in data structure $C_i(A^l)$ or $R_i(A^l)$ on level $l+1$ contains a single point of the level l data structure. In the general case, an element p in a level l data structure contains a point p' in a level l' data structure, $l > l'+1$, if p contains some p'' on level $l'+1$ and p'' contains p'. With every point p we store the set of points $cont(p)$ that consists of all points in the level 0 data structure that p contains. The set $cont(p)$ allows us to output directly the points which are contained in a cell K_{ij} in a data structure A_t on level $l > 0$

Lemma 2 *Data structure A supports planar range reporting queries in time* $O(\log n/\log n \log n + k)$.

Proof: A query $[a,b]\times[c,d]$ is processed as follows. We find i_{min}, i_{max} and j_{min}, j_{max} such that $c_{i_{min}-1} < a < c_{i_{min}}$, $c_{i_{max}-1} < b < c_{i_{max}}$, $r_{j_{min}-1} < c < r_{j_{min}}$, and $r_{j_{max}-1} < d < r_{j_{max}}$, where r_i and c_j are coordinates of row and column borders. Using T_c and T_r, i_{min}, i_{max} and j_{min}, j_{max} can be found in $O(\log n/\log\log n)$ time. We distinguish between two cases: **1.** $i_{min} = i_{max}$ or $j_{min} = j_{max}$. **2.** $i_{max} > i_{min}$ and $j_{max} > j_{min}$. In the first case the query rectangle is contained in one row or column, and in the second case $[a,b]\times[c,d]$ intersects with more than one row and more than one column. A row $[r_{i-1}, r_i]$ is called *a marginal row*, if $r_{i-1} < c < r_i$ or $r_i > d > r_{i-1}$. A column $[c_{i-1}, c_i]$ is called *a marginal column*, if $c_{i-1} < a < c_i$ or $c_i > b > c_{i-1}$. The query can be answered as follows:

Case 1 $[a,b]\times[c,d]$ is contained in one row R_i or in one column C_j. Then the search continues in the data structure corresponding to R_i or C_j.

Case 2 $[a,b]\times[c,d]$ intersects with more than one row and more than one column, i.e., there are internal rectangles. Then the query can be answered by answering four three-sided queries for marginal rows and columns and reporting all elements from non-empty internal rectangles using A_t. Let $c_{i_{min}-1} < a < c_{i_{min}} < \ldots < c_{i_{max}-1} < b < c_{i_{max}}$ and $r_{j_{min}-1} < c < r_{j_{min}} < \ldots < r_{j_{max}-1} < d < r_{j_{max}}$. That is, $C_{i_{min}}$,$C_{i_{max}}$, and $R_{j_{min}}$, $R_{j_{max}}$ are respectively marginal columns and marginal rows. We answer three-sided range queries $(c_{j_{max}-1}, b]\times[c,d]$ and $[a, c_{j_{min}})\times[c,d]$ using the fusion priority trees for $C_{j_{max}}$ and $C_{j_{min}}$. Those queries can be answered in $O(\log n/\log\log n + k)$ time, and for every point p in the answer we report all points in $cont(p)$ i.e., all points that p contains. We report the points from marginal rows in the same way using the fusion priority trees for $R_{i_{min}}$ and $R_{i_{max}}$. Finally, we identify all non-empty rectangles K_{ij} using a two-dimensional query $[j_{min}, j_{max}-1]\times[i_{min}, i_{max}-1]$ to A_t (see Fig. 1).

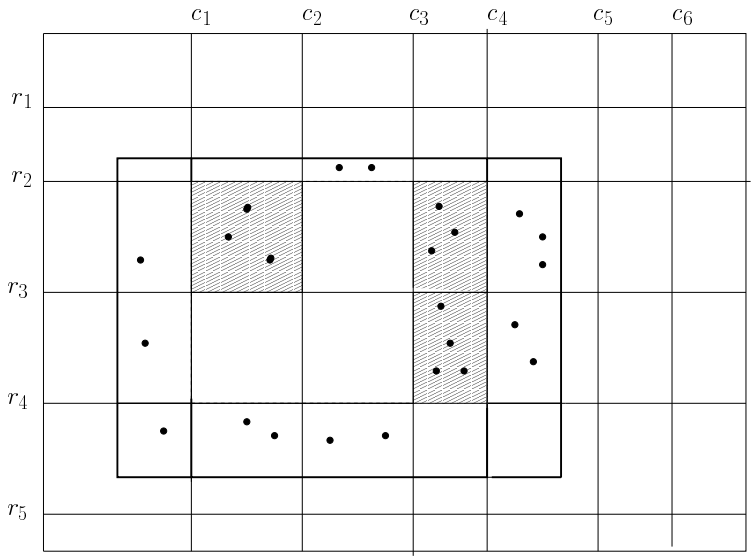

Figure 1: Range Query (Case 2). Three-sided range queries on marginal columns and rows are marked by bold lines. Non-empty K_{ij} are filled with slanted lines and can be identified by a query $[1,4] \times [2,4]$ to A_t.

Let $q(n,k)$ be query time. In the first case a query to a data structure with n elements is reduced to a query to a data structure with $O(n^{2/3})$ elements in $O(\log n/\log\log n)$ time. Thus $q(n,k) = O(\log n/\log\log n) + q(n^{2/3})$ in the first case. In the second case, a query is reduced to four three-sided queries to marginal columns and rows that can be answered in $O(\log n/\log\log n + k')$ time, and to query to a data structure A_t with $O(n^{2/3})$ elements that can be answered in $O(q(n^{2/3}, k''))$ time. Here k' denotes the number of points in marginal columns and rows, and k'' denotes the number of points in non-empty cells K_{ij}; $k' + k'' = k$. Observe that using A_t we can output the points in the level 0 data structure with help of $cont(p)$. Summing up, $q(n,k) \leq \max((q(n^{2/3}, k'') + O(\log n/\log\log n + k')), (q(n^{2/3}, k) + O(\log n/\log\log n)))$, where $k' + k'' = k$. Thus $q(n,k) \leq q(n^{2/3}, k') + O(\log n/\log\log n + k'')$ and $q(n,k) = O(\log n/\log\log n + k)$.

□

Lemma 3 *Data structure A uses $O(n\log^2 n)$ space and can be constructed in time $O(n\log^2 n)$*

Proof: Space $s(n)$ used by data structure A is superlinear because a polylogarithmic number of copies of each point must be stored. On the top level, every point must be stored in four fusion priority trees and two data structures with $O(n^{2/3})$ elements. Besides that, a data structure A_t with $O(n^{2/3})$ elements must be stored. We obtain a recursive formula $s(n) = O(n) + 2n^{1/3}s(n^{2/3}) + s(n^{2/3})$. Let $v(n) = s(n)/n$. Then

$v(n) = O(1) + 2v(n^{2/3}) + v(n^{2/3})/n^{1/3}$. Obviously, $v(n) < 3v(n^{2/3})$. Let $f(n) = v(2^n)$, then $f(n) < 3f((2/3)n)$, and $f(n) = O(n^3)$. Hence, $v(n) = O(\log^3 n)$ and $v(n) = 2v(n^{2/3}) + (O(1) + O(\log^3 n/n^{1/3})) = 2v(n^{2/3}) + o(n)$. Thus, $v(n) = O((\log n)^{\log_{3/2}(2)})$ and $v(n) = O(\log^2 n)$. Therefore, $s(n) = O(n \log^2 n)$.

Construction time $c(n)$ can be estimated with a recursive formula $c(n) = 4O(n) + 2n^{1/3}c(n^{2/3}) + c(n^{2/3})$. In the same way as above, $c(n) = O(n \log^2 n)$.

□

Lemma 4 *Update operations on the data structure A can be performed in $O(\log^a n)$ amortized time, where $a = \log_{3/2} 3 \approx 2.71$.*

Proof: Suppose an element e is inserted into or deleted from a data structure A of size n, and $e \in C_j(A)$, $e \in R_i(A)$ (e belongs to the j-th column and the i-th row). Then it must be inserted into or deleted from four fusion priority trees in time $O(\log n)$. Besides that, three data structures of size $O(n^{2/3})$, namely A_t, $C_j(A)$ and $R_i(A)$ for some i and j, must be modified; in case of A_t we either add/delete e from the set $cont(p)$, where p corresponds to K_{ij}, or an element p with coordinates (i, j) is added/deleted from A_t. The data structure can be updated in time $u(n) = u(n^{2/3}) + 2u(n^{2/3}) + O(\log n)$. Let $f(t) = u(2^t)$, then $f(t) = 3f(2t/3) + O(t)$. Applying master theorem, we obtain $f(t) = \Theta(t^{\log_{3/2} 3})$, and $u(n) = \Theta(\log n)^{\log_{3/2} 3} = \Theta(\log^a n)$.

In the above analysis we ignored the fact that some parts of the data structure must be rebuilt if certain conditions are violated (partial rebuild) and the whole data structure must be rebuilt sometimes (global rebuild). First consider the cost of rebuilding A (global rebuild). Suppose the number of elements after the last global rebuild was n_0. Then the next global rebuild takes place when $n_0/2 > |A|$ or $3n_0/2 < |A|$. During the global rebuilding the whole data structure is reconstructed "from scratch". As was shown in Lemma 3 this incurs the total cost of $O(n_0 \log^2 n_0)$ and the amortized cost $O(\log^2 n)$.

Now we estimate the amortized cost of local rebuilds. Consider some data structure A^l on level l containing m elements. Every row and column of A^l can hold between $m^{2/3}/2$ and $2m^{2/3}$ elements. If after a series of updates the number of elements in column C_i violates these bounds, we consider one of the neighbor columns C_{i+1} and C_{i-1}. Since $m^{2/3} \le |C_i \cup C_{i+1}| < 4m^{2/3}$ (the same bounds are also true for $|C_i \cup C_{i-1}|$), we can construct v columns from elements of C_i and C_{i+1}, where $1 \le v \le 4$, so that each new column contains between $3m^{2/3}/4$ and $3m^{2/3}/2$ elements. Since these columns can be rebuilt in time $O(m^{2/3} \log^2(m^{2/3}))$ (see Lemma 3) rebuilding columns incurs amortized cost $O(\log^2 m)$. Rebuilding rows is, of course, identical to rebuilding columns.

When a row or a column is rebuilt, A_t must also be updated. Namely, up to $O(m^{1/3})$ elements are inserted to or deleted from A_t. The total cost

of updating A_t can be estimated as $O(m^{2/3}\log^2 m)$ (i.e., instead of inserting $O(m^{1/3})$ elements, we rebuild the data structure with $O(m^{2/3})$ elements in $O(m^{2/3}\log^2 m)$ time. Thus updating A_t also incurs $O(\log^2 m)$ time.

The total amortized cost of rebuilds incurred by an insertion of a new element can be expressed as: $g(n) = O(\log^2 n) + 3g(n^{2/3})$. Substituting $f(n) = g(2^n)$ and solving the recurrence for $f(n)$, we obtain $g(n) = O(\log^a n)$. Thus the total amortized cost of an update operation is $O(\log^a n)$. □

2.1 A reduction from three-sided to planar range reporting queries

The result of Theorem 3 can be generalized as follows.

Theorem 4 *Suppose there is a data structure B for three-sided queries with query time $O(t(n)+k)$, where $t(n) = \Omega(\sqrt{\log n/\log\log n})$, and update time $u(n)$ that uses space $O(n)$ and can be constructed in $O(n)$ time. Then there exists a data structure A for planar range reporting queries that supports queries in time $O(t(n)+k)$ and updates in time $u'(n)$, such that $u'(n) = O(u(n)) + 3u'(n^{2/3})$; A uses space $O(n\log^2 n)$ and can be constructed in $O(n\log^2 n)$ time.*

Proof of Theorem 4 is analogous to the proof of Theorem 3.

3 A Space Efficient Data Structure

In this section we describe a general method for decreasing the space requirements of dynamic data structures for orthogonal range queries. The results described in the introduction follow from the combination of Theorem 5, Theorem 3, and Lemma 1.

Theorem 5 *Let $a = \log_{3/2} 3$, $t(n) = \Omega(\sqrt{\log n/\log\log n})$. Suppose there exist a data structure A for planar orthogonal range reporting queries with query time $O(t(n)+k)$ and update time $O(\log^a n)$ that requires space $O(n\log^p n)$ for any $p > 0$ and a linear space data structure B for three-sided queries with query time $O(t(n)+k)$ and update time $O(\log^a n)$.*
Then there is a data structure D that supports planar range reporting queries in $O(t(n)+k)$ time and requires $O(n\log^\varepsilon n)$ space for any $\varepsilon > 0$. There is also a data structure D' that supports planar range reporting queries in $O(t(n)+k\log\log n)$ time and requires $O(n\log\log n)$ space. Both D and D' support updates in amortized time $O(\log^a n)$. If A can be constructed in $O(n\log^p n)$ time, and B can be constructed in $O(n)$ time, then both D and D' can be constructed in $O(n\log n)$ time.

This theorem is a generalization of the result presented in [22] and can be proven in the same way. For completeness we provide a sketch of the

proof.

Proof Sketch : The set of points S is divided into columns C_i and rows R_i, so that the number of elements in every column (row) is between $\sqrt{n \log^p n}/2$ and $2\sqrt{n \log^p n}$. We store lists of points in all $K_{ij} = C_i \cap R_j$; data structure D_t contains points (i, j) for $K_{ij} \neq \emptyset$. Given A, D_t can be implemented in $O(n)$ space, so that queries and updates are supported in time $O(t(n) + k)$ and $O(\log^a n)$ respectively. For each column and row two data structures for three-sided range queries are stored. Using those data structures B, every query of the kind $(r_{i-1}, d] \times [a, b]$ or $[d, r_i) \times [a, b]$ or $(c_{j-1}, b] \times [c, d]$ and $[b, c_j) \times [c, d]$ can be answered in $O(t(n)+k)$ time. If the number of elements in a row or a column exceeds $t(n)$ a recursively defined data structure is stored for this row or column.

Consider a query $[a, b] \times [c, d]$. We find i_{min}, i_{max}, j_{min}, and j_{max}, such that $c_{i_{min}-1} < a$, $b < c_{i_{max}}$, $r_{j_{min}-1} < c$ and $d < r_{j_{max}}$. Again, we distinguish between two cases. If $c_{j-1} < a$ and $b < c_j$ for some j, or $r_{i-1} < c$ and $d < r_i$ for some i, then the query is transferred to a data structure corresponding to column C_j or row R_i. Otherwise we answer four three-sided range queries to report all elements from marginal columns and rows, and we identify all non-empty rectangles K_{ij} by answering query $[i_{min}, i_{max} - 1] \times [j_{min}, j_{max} - 1]$. In the first case, the size of the data structure is reduced from n to $\sqrt{n \log^p n}$, and in the second case the query is answered in $O(t(n) + k)$ time. Let query time $q(n, k) = O(t'(n) + k)$; $t'(n)$ can be estimated as $t'(n) = \max(O(t(n)), t'(\sqrt{n \log^p n}))$. $t'(n) < O(t(n)) + t'(\sqrt{n \log^p n})$. Since $\sqrt{n \log^p n} = o(n^{(b+1)/2b})$ for any integer $b > 0$, $t'(n) < O(t(n)) + t'(n^{(b+1)/2b})$. For $t(n) = \Omega(\sqrt{\log n / \log \log n})$, $t'(n) = O(t(n))$. Hence, queries can be answered in $O(t(n) + k)$ time.

We call the data structure that contains all points a *level 0 data structure*, and data structures that contain all points in a column or row of a level l data structure are called level $l + 1$ data structures. It can be shown that the maximal recursion level $l_{max} = \log \log n + c$, where c is a constant. Every point is stored in D $O(\log n)$ times: once in a level 0 data structure, twice in level 1 data structures, 2^l times in level l data structures. The space requirements can be reduced from $O(n \log n)$ to $O(n \log^\varepsilon n)$ for any $\varepsilon > 0$ by using the *dynamic range reduction to extended rank space* technique described in [22]. If we apply this technique on a recursion level l, every point in each level l data structure can be stored with only $O(\log(s^l(n)))$ bits, where $s^l(n)$ is the maximal number of points in a level l data structure. To avoid penalties for each point in the answer, we apply range reduction on every $\varepsilon \log \log n$-th level. Then every group of $\varepsilon \log \log n$ levels takes $O(n \log^{1+\varepsilon} n)$ bits, and the whole data structure D requires $O(n \log^\varepsilon n)$ words of $\log n$ bits.

Alternatively, to construct the data structure D', the dynamic range reduction can be applied on each recursive level. Then all data structures on level l, $l = 1, 2, \ldots, \log \log n + c$, use $O(n \log n)$ bits, and the total space can be reduced to $O(n \log \log n)$ words. But in this case the reverse range

reduction must be applied up to $\Theta(\log\log n)$ times to restore the original point coordinates, and for each point in the answer an $O(\log\log n)$ penalty must be paid. Therefore, the data structure D' uses $O(n\log\log n)$ words of memory, and supports queries in time $O(t(n)+k\log\log n)$.

Update time can be estimated with help of a recursive formula: $u(n) \leq O(\log^a n)+2u(\sqrt{n\log^p n}) < O(\log^a n)+2u(n^{(b+1)/2b})$ for an arbitrary integer $b > 1$. Therefore, $u(n) = O(\log^a n)$.

To construct D or D', we must construct a data structure A with $n/\log^p n$ elements, fusion priority trees for rows and columns, and $2\sqrt{n/\log^p n}$ data structures for rows and columns with $\sqrt{n\log^p n}$ elements each. Let $c'(n)$ be the construction time of A and $c(n)$ the construction time of D. Then for $c(n)$ a recursion $c(n) = c'(n/\log^p n) + 2\sqrt{n/\log^p n}c(\sqrt{n\log^p n}) + O(n)$ is valid. Let $v(n) = c(n)/n$. As $c'(n/\log^p n) = O(n)$, $v(n) = 2v(\sqrt{n\log^p n}) + O(1)$. Since the number of recursive levels is $\log\log n + c$ for a constant c, $v(n) = O(\log n)$, and $c(n) = O(n\log n)$. □

4 Conclusion

In this paper we presented a dynamic data structure for planar orthogonal range reporting with sublogarithmic query time.

Our results are based on two important reductions. Using those reductions a dynamic data structure for three-sided queries with query time $O(t(n)+k)$, where $t(n) = \Omega(\sqrt{\log n/\log\log n})$ (i.e., $t(n)$ is not asymptotically faster than the lower bound on the predecessor queries), can be converted into a data structure for general planar range reporting queries with only a small increase in space and without changing query time.

Several modifications of those reductions can be proven in a similar way. For instance, given a linear space data structure for three-sided queries on the $n \times n$ grid with query time $O(t(n)+k)$ for an arbitrary $t(n)$, a dynamic data structure for orthogonal range reporting on the $n \times n$ grid with query time $O(t'(n)+k)$, such that $t'(n) = O(t'(n^{2/3}) + t(n^{2/3}))$, and space $O(n\log^\varepsilon n)$ can be constructed. A similar reduction for data structures for three-sided queries with superlinear space can be also proven.

References

[1] P. K. Agarwal, L. Arge, A. Danner, B. Holland-Minkley "Cache-oblivious Data Structures for Orthogonal Range Searching."Proc. 9th ACM Symp. on Computational Geometry (2003), 237-245.

[2] P. K. Agarwal and J. Erickson "Geometric range searching and its relatives" In B. Chazelle, J. E. Goodman, and R. Pollack, ed-

itors, "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`.

[3] S. Alstrup, G. S. Brodal, T. Rauhe " New Data Structures for Orthogonal Range Searching", Proc. 41st IEEE FOCS(2000), 198-207.

[4] A. Andersson, *Faster Deterministic Sorting and Searching in Linear Space*, Proc 37th IEEE FOCS (1996), 135-141.

[5] A. Andersson, M. Thorup, *Tight(er) worst-case bounds on dynamic searching and priority queues*, Proc. 32nd ACM STOC(2000), 335-342.

[6] P. Beame, F. E. Fich, *Optimal Bounds for the Predecessor Problem and Related Problems*, J. Comput. Syst. Sci. vol. 65(1), 2002, 38-72.

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

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

[9] B. Chazelle, L. J. Guibas " Fractional Cascading: I. A Data Structuring Technique", Algorithmica, vol. 1(2), 1986, 133-162

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

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

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

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

[14] R. Klein, O. Nurmi, T. Ottman and D. Wood " A Dynamic Fixed Windowing Problem", Algorithmica vol. 4, 1989, 535-550.

[15] M. Van Kreveld and M.H. Overmars " Divided K-d Trees", Algorithmica vol. 6(6), 1991, 840-858.

[16] M. Van Kreveld and M.H. Overmars "Concatenable Structures for Decomposable Problems" Information and Computation, vol. 110(1), 1994, 130-148

[17] G. S. Lueker " A Data Structure for Orthogonal Range Queries", Proc. 19th ACM FOCS(1978), 28-34.

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

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

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

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

[22] Y. Nekrich, "Space efficient dynamic orthogonal range reporting", Proc. 21st ACM Symp. on Computational Geometry(2005), 306-313.

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

[24] M. H. Overmars "Efficient Data Structures for Range Searching on a Grid", J. Algorithms, vol. 9(2),1988, 254-275.

[25] R.E. Tarjan "A Class of Algorithms Which Require Nonlinear Time to Maintain Disjoint Sets", Journal of Computer and System Sciences vol. 18, 1979, 110-127.

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

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

[28] 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.

[29] D. E. Willard, Examining Computational Geometry, Van Emde Boas Trees, and Hashing from the Perspective of the Fusion Tree. SIAM J. Comput., vol. 29(3), 2000, 1030-1049.