Space-Efficient Data-Analysis Queries on Grids

Gonzalo Navarro[1] ⋆ and Luís M. S. Russo[2] ⋆⋆

[1] Dept. of Computer Science, University of Chile. gnavarro@dcc.uchile.cl
[2] INESC-ID / IST, Tech. Univ. of Lisbon, Portugal. luis.russo@ist.utl.pt

Abstract. We consider various data-analysis queries on two-dimensional points. We give new space/time tradeoffs over previous work on semigroup and group queries such as sum, average, variance, minimum and maximum. We also introduce new solutions to queries rarely considered in the literature such as two-dimensional quantiles, majorities, successor/predecessor and mode queries. We face static and dynamic scenarios.

1 Introduction

Multidimensional grids arise as natural representations to support conjunctive queries in databases [2]. Typical queries such as "find all the employees with age between x_0 and x_1 and salary between y_0 and y_1" translate into a two-dimensional range reporting query on coordinates age and salary.

Counting the points in a two-dimensional range $Q = [x_0, x_1] \times [y_0, y_1]$, *i.e.*, computing $\textsc{Count}(Q)$, is arguably the most primitive operation in *data analysis*. Given n points, one can compute $\textsc{Count}$ in time $O(\log n/\log\log n)$ using "linear" space, $O(n)$ *integers* [16]. This time is optimal within space $O(n \operatorname{polylog}(n))$ [19]. Bose *et al.* [3] achieve the same time using asymptotically minimum space, $n + o(n)$ integers.

In this paper we focus on other data-analysis queries. Our results build on the *wavelet tree* [11], a succinct-space variant of a classical structure by Chazelle [6]. The wavelet tree has been used to handle various geometric problems, *e.g.* [13, 3]. We adapt this structure for queries we call "statistical": The points have an associated value in $[0, W[= [0, W-1]$. Then, given a rectangle Q, we consider the following queries:

$\textsc{Sum}/\textsc{Avg}/\textsc{Var}$: The sum/average/variance of the values in Q.
$\textsc{Min}/\textsc{Max}$: The minimum/maximum value in Q.
$\textsc{Quantile}$: The k-th smallest value in Q.
$\textsc{Majority}(\alpha)$: The values appearing with relative frequency $> \alpha$ in Q.
$\textsc{Succ}/\textsc{Pred}$: The successor/predecessor of a value w in Q.

⋆ Supported in part by Millennium Institute for Cell Dynamics and Biotechnology (ICDB), Grant ICM P05-001-F, Mideplan, Chile.
⋆⋆ Funded by FCT proj. TAGS PTDC/EIA-EIA/112283/2009, ISTRION PTDC/EIA-EIA/114521/2009 and PIDDAC Program funds (INESC-ID multiannual funding).

These operations enable data-analysis queries such as “the average salary of employees whose annual production is between x_0 and x_1 and whose age is between y_0 and y_1”. The minimum operation can be used to determine “the employee with the lowest salary”, in the previous conditions. The α-majority operation can be used to compute “which are frequent ($\geq 20\%$) salaries”. Quantile queries are useful to determine “which are the 10% highest salaries”. Successor queries allow one determining “which are the salaries above \$100,000”.

Other applications for such queries are frequently found in Geographic Information Systems (GIS), where the points have a geometric interpretation and the values can be city sizes, industrial production, topographic heights, and so on. Yet another application comes from Bioinformatics, where two-dimensional points with intensities are obtained from DNA microarrays, and various kinds of data-analysis activities are carried out on them. See Rahul *et al.* [20] for an ample discussion on some of these applications and several others.

Willard [21] solved two-dimensional range-sum queries on finite groups within $O(n \log n)$-integers space and $O(\log n)$ time. This is easily extended to SUM, AVG, and VAR. Alstrup *et al.* [1] obtained the same complexities for the semigroup model, which includes MIN/MAX. The latter can also be solved in constant time using $O(n^2)$-integers space [4]. Rahul *et al.* [20] considered a variant of QUANTILE where one reports the *top-k smallest/largest values* in a range. They obtain $O(n \log^2 n)$-integers space and $O(\log n + k \log \log n)$ time. Durocher and Morrison [8] consider the *mode* (most repeated value) in a two-dimensional range. Their times are sublinear but super-polylogarithmic by far.

Section 3 gives our new results for statistical queries. Our spaces range from $O(n)$ to $O(n \log n)$ integers, offering novel space/time tradeoffs for partial sums on groups (including SUM, AVG, and VAR) and for MIN/MAX queries. We enrich wavelet trees with data that speeds up the computation of these queries. Space is then reduced by sparsifying this extra data. This gives in particular a space/time tradeoff to the recent top-k smallest/largest solution [20].

The solutions to quantile, majority and successor/predecessor queries use a single data structure, a wavelet tree built on the universe of the point values. A sub-grid at each node stores the points whose values are within a value range. With this structure we also solve mode and *top-k most-frequent* queries.

While we rarely improve the best current query times, we offer significant space reductions. This can be critical to maintain large datasets in main memory.

2 Wavelet Trees

Wavelet trees [11] are defined on top of the basic RANK and SELECT functions. Let B denote a bitmap, *i.e.,* a sequence of 0’s and 1’s. $\text{RANK}(B, b, i)$ counts the number of times bit $b \in \{0, 1\}$ appears in $B[0, i]$, assuming $\text{RANK}(B, b, -1) = 0$. The dual operation, $\text{SELECT}(B, b, i)$, returns the position of the i-th occurrence of b, assuming $\text{SELECT}(B, b, 0) = -1$.

The wavelet tree represents a sequence $S[0, n[$ over alphabet $\Sigma = [0, \sigma[$, and supports access to any $S[i]$, as well as RANK and SELECT on S, by reducing them to bitmaps. It is a complete binary tree where each node v may have a left child labeled 0 (called the 0-child of v) and a right child labeled 1 (called the 1-child). The sequence of labels obtained when traversing the tree from the ROOT down to a node v is the *binary label* of v and is denoted $L(v)$. Likewise we denote $V(L)$ the node that is obtained by following the sequence of bits L, thus $V(L(v)) = v$. The binary labels of the leaves correspond to the binary representation of the symbols of Σ. Given $c \in \Sigma$ we denote by $V(c)$ the leaf that corresponds to symbol c. By $c\{..d\}$ we represent the sequence of the first d bits in c. Therefore, for increasing values of d, the $V(c\{..d\})$ nodes represent the path to $V(c)$.

Each node v represents (but does not store) the subsequence $S(v)$ of S formed by the symbols whose binary code starts with $L(v)$. At each node v we only store a (possibly empty) bitmap, denoted $B(v)$, of length $|S(v)|$, so that $B(v)[i] = 0$ iff $S(v)[i]\{..d\} = L(v) \cdot 0$, where $d = |L(v)| + 1$, that is, if $S(v)[i]$ belongs to the 0-child. A bit position i in $B(v)$ can be mapped to a position in each of its child nodes: we map i to position $R(v, b, i) = \text{RANK}(B(v), b, i) - 1$ of the b-child. We refer to this procedure as the *reduction* of i, and use the same notation to represent a sequence of steps, where b is replaced by a sequence of bits. Thus $R(\text{ROOT}, c, i)$, for a symbol $c \in \Sigma$, represents the reduction of i from the ROOT using the bits in the binary representation of c. With this notation we describe the way in which the wavelet tree computes RANK, which is summarized by the equation $\text{RANK}(S, c, i) = R(\text{ROOT}, c, i) + 1$. We use a similar notation $R(v, v', i)$, to represent descending from node v towards a given node v', instead of explicitly describing the sequence of bits b such that $L(v') = L(v) \cdot b$ and writing $R(v, b, i)$.

An important path in the tree is the one obtained by choosing $R(v, B(v)[i], i)$ at each node, *i.e.,* at each node we decide to go left of right depending on the bit we are currently tracking. The resulting leaf is $V(S[i])$, therefore this process provides a way to obtain the elements of S. The resulting position is $R(\text{ROOT}, S[i], i) = \text{RANK}(S, S[i], i) - 1$.

It is also possible to move upwards on the tree, reverting the process computed by R. Let node v be the b-child of v'. Then, if i is a bit position in $B(v)$, we define the position $Z(v, v', i)$, in $B(v')$, as $\text{SELECT}(B(v'), b, i + 1)$. In general when v' is an ancestor of v the notation $Z(v, v', i)$ represents the iteration of this process. For a general sequence, SELECT can be computed by this process, as summarized by the equation $\text{SELECT}(S, c, i) = Z(V(c), \text{ROOT}, i - 1)$.

Lemma 1 ([11, 13, 18]). *The wavelet tree for a sequence $S[0, n[$ over alphabet $\Sigma = [0, \sigma[$ requires at most $n \log \sigma + o(n)$ bits of space.*[3] *It solves* RANK, SELECT, *and access to any $S[i]$ in time $O(\log \sigma)$.*

Proof. The structure proposed by Grossi *et al.* [11] used $n \log \sigma + O(\frac{n \log \sigma \log \log n}{\log n})$ $+O(\sigma \log n)$ bits. Mäkinen and Navarro showed how to use only one pointer per level, reducing the last term to $O(\log \sigma \log n) = O(\log^2 n) = o(n)$. Finally, Pătraşcu [18] showed how to support binary RANK and SELECT in constant time, while reducing the redundancy of the bitmaps to $O(n/\log^2 n)$, which added over the $n \log \sigma$ bits gives $o(n)$ as well. □

[3] From now on the space will be measured in bits and log will be to the base 2.

2.1 Representation of Grids

Consider a set $\mathbf{P}$ of n distinct two-dimensional points (x, y). We map coordinates to rank space using a standard method [6, 1]. We store two sorted arrays X and Y with all the (possibly repeated) x and y coordinates, respectively. Then we convert any point (x, y) into rank space $[0, n[\times[0, n[$ in time $O(\log n)$ using two binary searches. The space of X and Y corresponds to the bare point data and will not be further mentioned. Range queries are also mapped to rank space via binary searches (in an inclusive manner in case of repeated values). This mapping time will be dominated by other query times.

Therefore we store the points of $\mathbf{P}$ on a $[0, n[\times[0, n[$ grid, with exactly one point per row and one per column. We regard this set as a sequence $S[0, n[$ and the grid is formed by the points $(i, S[i])$. We represent S using a wavelet tree.

The information relative to a point $p_0 = (x_0, y_0)$ is usually tracked from the $\textsc{Root}$ and denoted $R(\textsc{Root}, y_0\{..d\}, x_0)$. A pair of points $p_0 = (x_0, y_0)$ and $p_1 = (x_1, y_1)$, where $x_0 \leq x_1$ and $y_0 \leq y_1$, defines a rectangle; this is the typical query range we consider in this paper. Rectangles have an implicit representation in wavelet trees, spanning $O(\log n)$ nodes [13]. The binary representation of y_0 and y_1 share a (possibly empty) common prefix. Therefore the paths $V(y_0\{..d\})$ and $V(y_1\{..d\})$ have a common initial path and then split at some node of depth k, *i.e.,* $V(y_0\{..d\}) = V(y_1\{..d\})$ for $d \leq k$ and $V(y_0\{..d'\}) \neq V(y_1\{..d'\})$ for $d' > k$. Geometrically, $V(y_0\{..k\}) = V(y_1\{..k\})$ corresponds to the smallest horizontal band of the form $[j \cdot n/2^k, (j+1) \cdot n/2^k[$ that contains the query rectangle Q, for an integer j. For $d' > k$ the nodes $V(y_0\{..d'\})$ and $V(y_1\{..d'\})$ correspond respectively to successively thinner, non-overlapping bands that contain the coordinates y_0 and y_1.

Given a rectangle $Q = [x_0, x_1] \times [y_0, y_1]$ we consider the nodes $V(y_0\{..d\} \cdot 1)$ such that $y_0\{..d\} \cdot 1 \neq y_0\{..d+1\}$, and the nodes $V(y_1\{..d\} \cdot 0)$ such that $y_1\{..d\} \cdot 0 \neq y_1\{..d+1\}$. These nodes, together with $V(y_0)$ and $V(y_1)$, form the implict representation of $[y_0, y_1]$, denoted $\textsc{imp}(y_0, y_1)$. The size of this set is $O(\log n)$. Let us recall a well-known application of this decomposition.

Lemma 2. *Given n two-dimensional points, the number of points inside a query rectangle $Q = [x_0, x_1] \times [y_0, y_1]$, $\textsc{Count}(Q)$, can be computed in time $O(\log n)$ with a structure that requires $n \log n + o(n)$ bits.*

Proof. The result is $\sum_{v \in \textsc{imp}(y_0, y_1)} R(\textsc{Root}, v, x_1) - R(\textsc{Root}, v, x_0 - 1)$. Notice that all the values $R(\textsc{Root}, y_0\{..d\}, x)$ and $R(\textsc{Root}, y_1\{..d\}, x)$ can be computed sequentially, in total time $O(\log n)$, for $x = x_1$ and $x = x_0 - 1$. For a node $v \in \textsc{imp}(y_0, y_1)$ the desired difference can be computed from one of these values in time $O(1)$. Then the lemma follows. $\square$

2.2 Dynamism

We can support point insertions and deletions on a fixed $n \times n$ grid. Dynamic variants of the bitmaps stored at each wavelet tree node raise the extra space to $o(\log n)$ per point and multiply the times by $O(\log n/ \log \log n)$ [12, 15].

Lemma 3. *Given s points on an $n \times n$ grid, there is a structure using $s \log n + o(s \log n)$ bits, answering queries in time $O(t \log n / \log \log n)$, where t is the time complexity of the query using static wavelet trees. It handles point insertions and deletions in time $O(\log^2 n / \log \log n)$.*

Proof. We use the same data structure and query algorithms of the static wavelet trees described in Section 2.1, yet representing their bitmaps with the dynamic variants [12, 15]. We also maintain vector X, but not Y; we use the y-coordinates directly instead since the wavelet tree handles repetitions in y.

Instead of an array, we use for X a multiary tree with arity $O(\sqrt{\log n})$ and with leaves holding $\Theta(\log n / \log \log n)$ elements [15]. This retains the same time penalty factor and poses an extra space (on top of the bare coordinates) of $o(s \log n)$ bits. When inserting a new point (x, y), apart from inserting x into X, we track the point downwards in the wavelet tree, doing the insertion at each of the $\log n$ bitmaps. Deletion is analogous. $\square$

3 Statistical Queries

We now consider points weighted by an integer-valued function $w : \mathbf{P} \to [0, W[$. Let us define the sequence of weights $W(v)$ associated to each wavelet tree node v. If $S(v) = p_0, p_1, \ldots, p_{|S(v)|}$, then $W(v) = w(p_0), w(p_1), \ldots, w(p_{|S(v)|})$.

3.1 Range Sum, Average, and Variance

We start with a solution to several range sum problems on groups.

Theorem 1. *Given n two-dimensional points with associated values in $[0, W[$, the sum of the point values inside a query rectangle $Q = [x_0, x_1] \times [y_0, y_1]$, $\textsc{Sum}(Q)$, can be computed in time $O(\ell \log_\ell n)$, with a structure that requires $n \log n + n \log_\ell n (\log W + O(1))$ bits, for any $\ell \in [2, n]$. It can also compute the average of the values, $\textsc{Avg}(Q)$. A similar structure requiring 3 times the space computes the variance of the values, $\textsc{Var}(Q)$.*

Proof. We enrich the bitmaps of the wavelet tree for $\mathbf{P}$. For each node v we represent its vector $W(v) = w(p_0), w(p_1), \ldots, w(p_{|S(v)|})$ as a bitmap $A(v)$, where we concatenate the unary representation of the $w(p_i)$'s, *i.e.,* $w(p_i)$ 0's followed by a 1. These bitmaps $A(v)$ are represented in a compressed format [17] which requires at most $|S(v)| \log W + O(|S(v)|)$ bits. With this structure we can determine the sum $w(p_0) + w(p_1) + \ldots + w(p_i)$, *i.e.,* the partial sums, in constant time by means of $\textsc{Select}(A(v), 1, i)$ queries[4], $\textsc{Wsum}(v, i) = \textsc{Select}(A(v), 1, i+1) - i$ is the sum of the first $i + 1$ values. In order to compute $\textsc{Sum}(Q)$ we use a formula similar to the one of Lemma 2:

$$\sum_{v \in \textsc{imp}(y_0, y_1)} \textsc{Wsum}(v, R(\textsc{Root}, v, x_1)) - \textsc{Wsum}(v, R(\textsc{Root}, v, x_0 - 1)). \qquad (1)$$

[4] Using constant-time $\textsc{Select}$ structures on their internal bitmap H [17].

To obtain the tradeoff related to ℓ, we store the A bitmaps only for nodes v whose height $h(v)$ is a multiple of $h = \lceil \log \ell \rceil$, including the leaves. If a node $v \in \textsc{imp}(y_0, y_1)$ occurs at a level that does not have an associated bitmap $A(v)$, it is necessary to branch recursively to both children until we reach a level that has such a bitmap. In this case we use Eq. (1) to sum over all the nodes in the levels that have $A(v)$ bitmaps. The time raises by up to $2^h = O(\ell)$ factor. However, if a node v must access $2^h > 1$ bitmaps $A(\cdot)$, a node v' with height $h(v') = h(v)+1$ must access only 2^{h-1}. As in $\textsc{imp}(y_0, y_1)$ there are at most two nodes of each height, the total extra cost over all the nodes of heights $[h \cdot i, h \cdot (i+1)[$, for any $i \in [0, \log(n)/h[$ is not $O(h\ell)$ but amortizes to $2^h + 2^{h-1} + \ldots = 2 \cdot 2^h$, and hence the overall extra cost factor is $O(2^h/h) = O(\ell/\log \ell)$.

The average weight inside Q is computed as $\textsc{Avg}(Q) = \textsc{Sum}(Q)/\textsc{Count}(Q)$, where the latter is computed with the same structure by just adding up the interval lengths in $\textsc{imp}(y_0, y_1)$. To compute variance we use an additional instance of the same data structure, with weights $w'(p) = w^2(p)$, and then $\textsc{Var}(Q) = \textsc{Sum}'(Q) - \textsc{Sum}(Q)^2/\textsc{Count}(Q)$, where $\textsc{Sum}'$ uses the weights w'. The space triples because the weights $w'(p)$ require $\lceil 2 \log W \rceil$ bits in bitmaps $A(v)$. □

The solution applies to finite groups $(G, \oplus, ^{-1}, 0)$. We store $\textsc{Wsum}(v, i) = w(p_0) \oplus w(p_1) \oplus \ldots w(p_i)$, directly using $\lceil \log |G| \rceil$ bits per entry. The terms $\textsc{Wsum}(v, i) - \textsc{Wsum}(v, j)$ of Eq. (1) are replaced by $\textsc{Wsum}(v, j)^{-1} \oplus \textsc{Wsum}(v, i)$.

A dynamic variant is obtained by using the dynamic wavelet trees of Lemma 3, and a dynamic partial sums data structure [14] instead of $A(v)$.

Theorem 2. *Given s points on an $n \times n$ grid, with associated values in $[0, W[$, there is a structure using $s \log n + s \log_\ell n \log W(1 + o(1))$ bits, for any $\ell \in [2, W]$, that answers the queries in Thm. 1 in time $O(\ell \log^2 n)$, and supports point insertions/deletions, and value updates, in time $O(\log^2 n / \log \log n + \log n \log_\ell n)$.*

Proof. The dynamic searchable partial sums [14] take $n \log W + o(n \log W)$ bits to store an array of n values, and support partial sums, as well as insertions and deletions of values, in time $O(\log n)$. This multiplies the time complexities of Thm. 1. For insertions we must insert the new bits at all the levels as in Lemma 3 and also insert the new values at $\log_\ell n$ levels. Deletions are analogous. To update a value we just delete and reinsert the point. □

3.2 Range Minima and Maxima

For the one-dimensional problem there exists a data structure using just $2n+o(n)$ bits, which answers queries in constant time without accessing the values [9]. This structure allows for a better space/time tradeoff compared to range sums.

For the queries that follow we do not need the exact $w(p)$ values, but just their order. So, if $W > n$, we store an array with the (at most) n different values of $w(p)$ in the point set, and map all the weights to rank space in time $O(\log n)$, which will be negligible. The extra space that this requires is $n \lceil \log W \rceil$ bits. In exchange, many complexities will be expressed in terms of $u = \min(n, W)$.

Theorem 3. *Given n two-dimensional points with associated values in $[0, W[$, the minimum of the point values inside a query rectangle $Q = [x_0, x_1] \times [y_0, y_1]$, $\textsc{Min}(Q)$, can be found in time $O(\log \ell \log n)$, with a structure using $n \log_\ell n \log u +$ $O(n \log n) + n \log W$ bits, for any $\ell \in [2, n]$ and $u = \min(n, W)$. A similar structure answers $\textsc{Max}(Q)$ within the same space and time.*

Proof. We associate to each node v the one-dimensional data structure [9] corresponding to $W(v)$, which takes $2|W(v)| + o(|W(v)|)$ bits. This adds up to $O(n \log n)$ bits overall. Let us call $\textsc{Wmin}(v, i, j) = \arg\min_{i \le m \le j} W(v)[m]$ the one-dimensional operation. Then we can find in constant time the position of the minimum value inside each $v \in \textsc{imp}(y_0, y_1)$ (without the need to store the values in the node), and the range minimum is

$$\min_{v \in \textsc{imp}(y_0, y_1)} W(v)[\textsc{Wmin}(v, R(\textsc{Root}, v, x_0), R(v, \textsc{Root}, v, x_1 + 1) - 1)].$$

To complete the comparison we need to compute the $O(\log n)$ values $W(v)[m]$ of different nodes v. By storing the $A(v)$ bitmaps of Thm. 1 every $\log \ell$ levels, the time is just $O(\log \ell \log n)$ because we have to track down just one point for each $v \in \textsc{imp}(y_0, y_1)$. The case of $\textsc{Max}(Q)$ is analogous. $\square$

We can now solve the top-k query of Rahul *et al.* [20] by iterating over Thm. 3. Once we identify that the overall minimum is some $W(v)[m]$ from the range $W(v)[i, j]$, we can find the second minimum among the other candidate ranges plus the ranges $W(v)[i, m-1]$ and $W(v)[m+1, j]$. A priority queue handling the ranges will perform k minimum extractions and $O(k + \log n)$ insertions, and its size will be limited to k. So the overall time is $O(\log \ell \log n + k(\log \ell + \log k))$ using a priority queue with constant insertion time [5]. This is comparable with previous time complexities [20] within much less space (even for $\ell = 1$).

3.3 Range Median and Quantiles

We compute the median element, or more generally, the k-th smallest value $w(p)$ in an area $Q = [x_0, x_1] \times [y_0, y_1]$ (the median corresponds to $k = \textsc{Count}(Q)/2$).

From now on we use a different wavelet tree decomposition, on the universe $[0, u[$ of $w(\cdot)$ values rather than on y coordinates. This can be seen as a wavelet tree on grids rather than on sequences: the node v of height $h(v)$ stores a grid $G(v)$ with the points $p \in \mathbf{P}$ such that $\lfloor w(p)/2^{h(v)} \rfloor = L(v\{..\lceil \log u \rceil - h(v)\})$. Note that each leaf c stores the points p with weight $w(p) = c$.

Theorem 4. *Given n two-dimensional points with associated values in $[0, W[$, the k-th smallest value of points within a query rectangle $Q = [x_0, x_1] \times [y_0, y_1]$, $\textsc{Quantile}(k, Q)$, can be found in time $O(\ell \log n \log_\ell u)$, with a structure using $n \log n \log_\ell u + O(n \log u) + n \log W$ bits, for any $\ell \in [2, u]$ and $u = \min(n, W)$. The time drops to $O(\ell \log n \log_\ell u / \log \log n)$ by using $n \log n \log_\ell u(1 + o(1)) +$ $O(n \log u) + n \log W$ bits of space.*

Proof. We use the wavelet tree on grids just described, representing each grid $G(v)$ with the structure of Lemma 2. To solve this query we start at root of the wavelet tree of grids and consider its left child, v. If $t = \textsc{Count}(Q) \geq k$ on grid $G(v)$, we continue the search on v. Otherwise we continue the search on the right child of the root, with parameter $k - t$. When we arrive at a leaf corresponding to value c, then c is the k-th smallest value in $\mathbf{P} \cap Q$.

Notice that we need to reduce the query rectangle to each of the grids $G(v)$ found in the way. We store the X and Y arrays only for the root grid, which contains the whole $\mathbf{P}$. For this and each other grid $G(v)$, we store a bitmap $X(v)$ so that $X(v)[i] = b$ iff the i-th point in x-order is stored at the b-child of v. Similarly, we store a bitmap $Y(v)$ with the same bits in y-order. Therefore, when we descend to the b-child of v, for $b \in \{0, 1\}$, we remap x_0 to $\textsc{Rank}(X(v), b, x_0)$ and x_1 to $\textsc{Rank}(X(v), b, x_1 + 1) - 1$, and analogously for y_0 and y_1 with $Y(v)$.

The bitmaps $X(v)$ and $Y(v)$ add up to $O(n \log u)$ bits of space. For the grids, consider that each point in each grid contributes at most $\log n + o(1)$ bits, and each $p \in \mathbf{P}$ appears in $\lceil \log u \rceil - 1$ grids (as the root grid is not really necessary).

To reduce space, we store the grids $G(v)$ only every $\lceil \log \ell \rceil$ levels (the bitmaps $X(v)$ and $Y(v)$ are still stored for all the levels). This gives the promised space. For the time, the first decision on the root requires computing up to ℓ operations $\textsc{Count}(Q)$, but the decision on its child requires half of them. Just as in Thm. 1, the total time adds up to $O(\ell \log n \log_\ell u)$.

To achieve the final alternative space/time, replace our wavelet trees of Lemma 2 by Bose et al.'s counting structure [3]. $\square$

Note that the basic construction allows us to count the number of points $p \in Q$ whose values $w(p)$ fall in a given range $[w_0, w_1]$, within time $O(\ell \log n \log_\ell u)$ or $O(\ell \log n \log_\ell u / \log \log n)$. This is another useful operation for data analysis, and can be obtained with the formula $\sum_{v \in \textsc{imp}(w_0, w_1)} \textsc{Count}(Q)$.

3.4 Range Majority

An α-majority of a rectangle $Q = [x_0, x_1] \times [y_0, y_1]$ is a weight that occurs more than $\alpha \cdot \textsc{Count}(Q)$ times inside Q, for some $\alpha \in [0, 1]$. We can solve this problem with the same structure used for Thm. 4.

Theorem 5. *The structures of Thm. 4 can compute all the α-majorities of the point values inside Q, $\textsc{Majority}(\alpha, Q)$, in time $O(\frac{1}{\alpha} \ell \log n \log_\ell u)$ or $O(\frac{1}{\alpha} \ell \log n \log_\ell u / \log \log n)$, respectively, where α can be chosen at query time.*

Proof. For $\alpha \geq \frac{1}{2}$ we find the median c of Q and then use the leaf c to count its frequency in Q. If this is more than $\alpha \cdot \textsc{Count}(Q)$, then c is the answer, else there is no α-majority. For $\alpha < \frac{1}{2}$, we solve the query by probing all the $(i \cdot \alpha)\textsc{Count}(Q)$-th elements in Q. $\square$

Culpepper *et al.* [7] show how to find the mode, and in general the k most repeated values inside Q, using successively more refined $\textsc{Quantile}$ queries. Let the k-th most repeated value occur $\alpha \cdot \textsc{Count}(Q)$ times in Q, then we require at most $4/\alpha$ quantile queries [7]. The same result can be obtained by probing successive values $\alpha = 1/2^i$ with $\textsc{Majority}(\alpha)$ queries.

3.5 Range Successor and Predecessor

The successor (predecessor) of a value w in a rectangle $Q = [x_0, x_1] \times [y_0, y_1]$ is the smallest (largest) weight larger (smaller) than, or equal to, w in Q. We also have an efficient solution using our wavelet trees on grids.

Theorem 6. *The structures of Thm. 4 can compute the successor and predecessor of a value w within the values of the points inside Q,* $\textsc{Succ}(w, Q)$ *and* $\textsc{Pred}(w, Q)$*, in time $O(\ell \log n \log_\ell u)$ or $O(\ell \log n \log_\ell u / \log\log n)$, respectively.*

Proof. We consider the nodes $v \in \textsc{imp}(w, +\infty)$ from left to right, tracking rectangle Q in the process. The condition for continuing the search below a node v that is in $\textsc{imp}(w, +\infty)$, or is a descendant of one such node, is that $\textsc{Count}(Q) > 0$ on $G(v)$. $\textsc{Succ}(w, Q)$ is the value associated with the first leaf found by this process. Likewise, $\textsc{Pred}(w, Q)$ is computed by searching $\textsc{imp}(-\infty, w)$ from right to left. To reduce space we store the grids only every $\lceil \log \ell \rceil$ levels, and thus determining whether a child has a point in Q may cost up to $O(\ell \log n)$. Yet, as for Thm. 1, the total time amortizes to $O(\ell \log n \log_\ell u)$. □

3.6 Dynamism

Our dynamic wavelet tree of Lemma 3 supports range counting and point insertions/deletions on a fixed grid in time $O(\log^2 n / \log\log n)$ (other tradeoffs exist [16]). If we likewise assume that our grid is fixed in Thms. 4, 5 and 6, we can also support point insertions and deletions (and thus changing the value of a point).

Theorem 7. *Given s points on an $n \times n$ grid, with associated values in $[0, W[$, there is a structure using $s \log n \log_\ell W(1+o(1))$ bits, for any $\ell \in [2, W]$, that answers queries* $\textsc{Quantile}$, $\textsc{Succ}$ *and* $\textsc{Pred}$ *in time $O(\ell \log^2 n \log_\ell W / \log\log n)$, and query* $\textsc{Majority}(\alpha)$ *in time $O(\frac{1}{\alpha} \ell \log^2 n \log_\ell W / \log\log n)$. It supports point insertions and deletions, and value updates, in time $O(\log^2 n \log_\ell W / \log\log n)$.*

Proof. We use the data structure of Thms. 4, 5 and 6, modified as follows. We build the wavelet tree on the universe $[0, W[$ and thus do not map the universe values to rank space. The grids $G(v)$ use the dynamic structure of Lemma 3, on global y-coordinates $[0, n[$. We maintain the global array X of Lemma 3 plus the vectors $X(v)$ of Thm. 4, the latter using dynamic bitmaps [12, 15]. The time for the queries follows immediately. For updates we track down the point to insert/delete across the wavelet tree, inserting or deleting it in each grid $G(v)$ found in the way, and also in the corresponding vector $X(v)$. □

4 Final Remarks

Many other data-analysis queries may be of interest. A prominent one lacking good solutions is to find the mode, that is, the most frequent value, in a rectangle, and its generalization to the top-k most frequent values. There has been some recent progress on the one-dimensional version [10] and even in two dimensions [8], but the results are far from satisfactory.

References

1. S. Alstrup, G. Brodal, and T. Rauhe. New data structures for orthogonal range searching. In *FOCS*, pages 198–207, 2000.
2. M. Berg, O. Cheong, M. Kreveld, and M. Overmars. Orthogonal range searching: Querying a database. In *Computational Geometry*, pages 95–120. Springer, 2008.
3. P. Bose, M. He, A. Maheshwari, and P. Morin. Succinct orthogonal range search structures on a grid with applications to text indexing. In *WADS*, pages 98–109, 2009.
4. G. Brodal, P. Davoodi, and S. Rao. On space efficient two dimensional range minimum data structures. *Algorithmica*, 2011. DOI 10.1007/s00453-011-9499-0.
5. S. Carlsson, J. I. Munro, and P. V. Poblete. An implicit binomial queue with constant insertion time. In *SWAT*, pages 1–13, 1988.
6. B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comp.*, 17(3):427–462, 1988.
7. S. Culpepper, G. Navarro, S. Puglisi, and A. Turpin. Top-k ranked document search in general text databases. In *ESA*, pages 194–205 (part II), 2010.
8. S. Durocher and J. Morrison. Linear-space data structures for range mode query in arrays. *CoRR*, 1101.4068, 2011.
9. J. Fischer. Optimal succinctness for range minimum queries. In *LATIN*, pages 158–169, 2010.
10. T. Gagie, G. Navarro, and S. Puglisi. Colored range queries and document retrieval. In *SPIRE*, pages 67–81, 2010.
11. R. Grossi, A. Gupta, and J. Vitter. High-order entropy-compressed text indexes. In *SODA*, pages 841–850, 2003.
12. M. He and I. Munro. Succinct representations of dynamic strings. In *SPIRE*, pages 334–346, 2010.
13. V. Mäkinen and G. Navarro. Rank and select revisited and extended. *Theo. Comp. Sci.*, 387(3):332–347, 2007.
14. V. Mäkinen and G. Navarro. Dynamic entropy-compressed sequences and full-text indexes. *ACM Trans. Alg.*, 4(3):art. 32, 2008.
15. G. Navarro and K. Sadakane. Fully-functional static and dynamic succinct trees. *CoRR*, 0905.0768v5, 2010.
16. Y. Nekrich. Orthogonal range searching in linear and almost-linear space. *Comp. Geom. Theo. and App.*, 42(4):342–351, 2009.
17. D. Okanohara and K. Sadakane. Practical entropy-compressed rank/select dictionary. In *ALENEX*, 2007.
18. M. Pătraşcu. Succincter. In *FOCS*, pages 305–313, 2008.
19. M. Pătraşcu. Lower bounds for 2-dimensional range counting. In *STOC*, pages 40–46, 2007.
20. S. Rahul, P. Gupta, R. Janardan, and K. Rajan. Efficient top-k queries for orthogonal ranges. In *WALCOM*, pages 110–121, 2011.
21. D. Willard. New data structures for orthogonal range queries. *SIAM J. Comp*, 14(232-253), 1985.