Theoretical Computer Science 482 (2013) 60–72

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Space-efficient data-analysis queries on grids

Gonzalo Navarro [a], Yakov Nekrich [a], Luís M.S. Russo [c,b,*]

[a] Department of Computer Science, University of Chile, Chile

[b] KDBIO/INESC-ID, Portugal

[c] Instituto Superior Técnico, Universidade Técnica de Lisboa (IST/UTL), Lisboa, Portugal

ARTICLE INFO

Article history:
Received 21 December 2011
Received in revised form 1 August 2012
Accepted 27 November 2012
Communicated by G. Italiano

Keywords:
Range queries
Databases
Succinct data structures
Dynamic data structures
Statistical database queries
Orthogonal range queries
Point dominance
Rectangle visibility
Wavelet tree
Range minimum queries
Alpha majority
Quantile
Top-*k*
Mode

ABSTRACT

We consider various data-analysis queries on two-dimensional points. We give new space/time tradeoffs over previous work on geometric queries such as dominance and rectangle visibility, and on semigroup and group queries such as sum, average, variance, minimum and maximum. We also introduce new solutions to queries less frequently considered in the literature such as two-dimensional quantiles, majorities, successor/predecessor, mode, and various top-*k* queries, considering static and dynamic scenarios.

1. Introduction

Multidimensional grids arise as natural representations to support conjunctive queries in databases [1]. 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. More generally, such a grid representation of the data is useful to carry out a number of *data analysis* queries over large repositories. Both space and time efficiency are important when analyzing the performance of data structures on massive data. However, in cases of very large data volumes the space usage can be even more important than the time needed to answer a query. More or less space usage can make the difference between maintaining all the data in main memory or having to resort to disk, which is orders of magnitude slower.

In this paper we study various problems on two-dimensional grids that are relevant for data analysis, focusing on achieving good time performance (usually polylogarithmic) within the least possible space (even succinct for some problems).

* Corresponding author at: Instituto Superior Técnico, Universidade Técnica de Lisboa (IST/UTL), Lisboa, Portugal. Tel.: +351 21 31 00272; fax: +351 21 31 45843.

E-mail addresses: gnavarro@dcc.uchile.cl (G. Navarro), yakov.nekrich@googlemail.com (Y. Nekrich), luis.russo@ist.utl.pt (L.M.S. Russo).

0304-3975/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2012.11.031

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 on an $n \times n$ grid, one can compute $\textsc{Count}$ in time $O(\log n/\log\log n)$ using "linear" space, $O(n)$ *integers* [2]. This time is optimal within space $O(n \operatorname{polylog}(n))$ [3], and it has been matched using asymptotically minimum (*i.e.,* succinct) space, $n + o(n)$ integers, by Bose et al. [4].

The k points in the range can be reported in time $O(\log\log n + k)$ using $O(n \log^{\epsilon} n)$ integers, for any constant $\epsilon > 0$ [5]. This time is optimal within space $O(n \operatorname{polylog}(n))$, by reduction from the colored predecessor problem [6]. With $O(n)$ integers, the time becomes $O((k+1)\log^{\epsilon} n)$ [7]. Within $n + o(n)$ integers space, one can achieve time $O(k\frac{\log n}{\log\log n})$ [4].

We start with two geometric problems, *dominance* and *rectangular visibility*. These enable data analysis queries such as "find the employees with worst productivity–salary combination within productivity range $[x_0, x_1]$ and salary range $[y_0, y_1]$", that is, such that no less productive employee earns more.

The best current result for the 4-sided variant of these problems (*i.e.,* where the points are limited by a general rectangle Q) is a dynamic structure by Brodal and Tsakalidis [8]. It requires $O(n \log n)$-integers space and reports the d dominant/visible points in time $O(\log^2 n + d)$. Updates take $O(\log^2 n)$ time. They achieve better complexities for simpler variants of the problem, such as some 3-sided variants.

Our results build on the *wavelet tree* [9], a succinct-space variant of a classical structure by Chazelle [10]. The wavelet tree has been used to handle various geometric problems, *e.g.,* [11,12,4,13–15]. We show in Section 3 how to use wavelet trees to solve dominance and visibility problems using $n + o(n)$ integers space and $O((d+1)\log n)$ time. The dynamic version also uses succinct space and requires $O((d+1)\log^2 n/\log\log n)$ time, carrying out updates in time $O(\log^2 n/\log\log n)$. Compared to the best current result [8], our structure requires succinct space instead of $O(n \log n)$ integers, offers better update time, and has a comparable query time (being faster for small $d = O(\log\log n)$).

The paper then considers a wide range of queries we call "statistical": The points have an associated value in $[0, W) = [0, W-1]$ and, given a rectangle Q, we consider the following queries:

$\textsc{Sum/Avg/Var}$: The sum/average/variance of the values in Q (Section 4).
$\textsc{Min/Max}$: The minimum/maximum value in Q (Section 5).
$\textsc{Quantile}$: The k-th smallest value in Q (Section 7).
$\textsc{Majority}(\alpha)$: The values appearing with relative frequency $> \alpha$ in Q (Sections 6 and 8).
$\textsc{Succ/Pred}$: The successor/predecessor of a value w in Q (Section 9).

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 enable us to determine "which are the 10% highest salaries". Successor queries can be used to find the smallest salary over \$100,000 among those employees.

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. [16] for an ample discussion on some of these applications and several others.

A popular set of statistical queries includes range sums, averages, and maxima/minima. Willard [17] solved two-dimensional range-sum queries on finite groups within $O(n \log n)$-integers space and $O(\log n)$ time. This includes $\textsc{Sum}$ and is easily extended to $\textsc{Avg}$ and $\textsc{Var}$. Alstrup et al. [5] obtained the same complexities for the semigroup model, which includes $\textsc{Min/Max}$. The latter can also be solved in constant time using $O(n^2)$-integers space [18,19]. Chazelle [10] showed how to reduce the space to $O(n/\epsilon)$ integers and achieve time $O(\log^{2+\epsilon} n)$ on semigroups, and $O(\log^{1+\epsilon} n)$ for the particular case of $\textsc{Min/Max}$.

On this set of queries our contribution is to achieve good time complexities within linear and even succinct space. This is relevant to handle large datasets in main memory. While the times we achieve are not competitive when using $O(n)$ integers or more, we manage to achieve polylogarithmic times within just $n \log n + o(n \log n)$ bits on top of the bare coordinates and values, which we show is close to the information-theoretic minimum space necessary to represent n points.

As explained, we use wavelet trees. These store bit vectors at the nodes of a tree that decomposes the y-space of the grid. The vectors track down the points, sorted on top by x-coordinate and on the bottom by y-coordinate. We enrich wavelet trees with extra data aligned to the bit vectors, which speeds up the computation of the statistical queries. Space is then reduced by sparsifying these extra data.

We also focus on more sophisticated queries, for which fewer results exist, such as quantile, majority, and predecessor/successor queries.

In one dimension, the best result we know of for quantiles queries is a linear-space structure by Brodal and Jørgensen [20], which finds the k-th element of any range in an array of length n in time $O(\log n/\log\log n)$, which is optimal.

An α-majority of a range Q is a value that occurs more than $\alpha \cdot \textsc{Count}(Q)$ times inside Q, for some $\alpha \in [0, 1]$. The α-majority problem was previously considered in one and two dimensions [21–23]. Durocher et al. [22] solve one-dimensional α-range majority queries in time $O(1/\alpha)$ using $O(n(1+\log(1/\alpha)))$ integers. Here α must be chosen when creating the data structure. A more recent result, given by Gagie et al. [23], obtains a structure of $O(n^2(H+1)\log(1/\alpha))$ bits for a dense $n \times n$

Table 1
Our static results on statistical queries, for n two-dimensional points with associated values in $[0, W)$; $m = \min(n, W)$; $2 \le \ell \le u$ and $t \ge 1$ are parameters. The space omits the mapping of the real (x, y) coordinates to the space $[0, n)$, as well as the storage of the point values. The 4th column gives simplified time assuming $\log n = \Theta(\log W)$, any constant ϵ, and use of $O(n \log n)$ bits.

Operation	Space per point (bits)	Time	Time in linear space	Source
Sum, Avg, Var	$\log n(1 + 1/t)$	$O(\min(t \log W, \log n)t \log W \log n)$	$O(\log^3 n)$	Theorem 6
Min, Max	$\log n(1 + 1/t)$	$O(\min(t \log m, \log n)t \log n)$	$O(\log^2 n)$	Theorem 8
Majority(α), fixed	$\log n(2 + 1/t) + \log m$	$O(t \log m \log^2 n)$	$O(\log^3 n)$	Theorem 10
Quantile	$\log n \log_\ell m + O(\log m)$	$O(\ell \log n \log_\ell m)$	$O(n^\epsilon)$	Theorem 11
Majority(α), variable	$\log n \log_\ell m + O(\log m)$	$O(\frac{1}{\alpha} \ell \log n \log_\ell m)$	$O(n^\epsilon)$	Theorem 12
Succ, Pred	$\log n \log_\ell m + O(\log m)$	$O(\ell \log n \log_\ell m)$	$O(n^\epsilon)$	Theorem 13

Table 2
Our dynamic results on statistical queries, for n two-dimensional points on an $U \times U$ grid with associated values in $[0, W)$; $2 \le \ell \le W$, $t \ge 1$ and $0 < \epsilon < 1$ are parameters. The space omits the mapping of the real x coordinates to $[0, n)$, as well as the point values. The first line is proved in Theorem 7, the second in Theorem 9, and the rest in Theorem 14.

Operation	Space per point (bits)	Query time	Update time
Sum, Avg, Var	$\log U(1 + o(1) + 1/t)$	$O(\log U \log n(1 + \frac{\min(t \log W, \log U)t \log W}{\log \log n}))$	$O(\log U \log n)$
Min, Max	$\log U(1 + o(1) + 1/t)$	$O(\log U \log n(1 + \frac{\min(t \log W, \log U)t \log W}{\log \log n}))$	$O(\log U(\log n + t \log W))$
Quantile	$\log U \log_\ell W(1 + o(1))$	$O(\ell \log U \log n \log_\ell W / \log \log n)$	$O(\log U \log n \log_\ell W / \log \log n)$
Majority(α), var.	$\log U \log_\ell W(1 + o(1))$	$O(\frac{1}{\alpha} \ell \log U \log n \log_\ell W / \log \log n)$	$O(\log U \log n \log_\ell W / \log \log n)$
Succ, Pred	$\log U \log_\ell W(1 + o(1))$	$O(\ell \log U \log n \log_\ell W / \log \log n)$	$O(\log U \log n \log_\ell W / \log \log n)$

matrix (*i.e.,* every position contains an element), where H is the entropy of the distribution of elements. In this case α is also chosen at indexing time, and the structure can answer queries for any $\beta \ge \alpha$. The resulting elements are not guaranteed to be β-majorities, as the list may contain false positives, but there are no false negatives.

Other related queries have been studied in two dimensions. Rahul et al. [16] 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. Navarro and Nekrich [24] reduced the space to $O(n/\epsilon)$ integers, with time $O(\log^{1+\epsilon} n + k \log^\epsilon n)$. Durocher and Morrison [25] consider the *mode* (most repeated value) in a two-dimensional range. Their times are sublinear but super-polylogarithmic by far.

Our contribution in this case is a data structure of $O(n \log n)$ integers able to solve the three basic queries in time $O(\log^2 n)$. The space can be stretched up to linear, but at this point the times grow to the form $O(n^\epsilon)$. Our solution for range majorities lets α to be specified at query time. For the case of α known at indexing time, we introduce a new linear-space data structure that answers queries in time $O(\log^3 n)$, and up to $O(\log^2 n)$ when using $O(n \log n)$ integers space.

In this case we build a wavelet tree on the universe of the point values. A sub-grid at each node stores the points whose values are within a range. With this structure we can also solve mode and *top-k most-frequent* queries.

Table 1 shows the time and space results we obtain in this article for statistical queries. Several of our data structures can be made dynamic at the price of a sublogarithmic penalty factor in the time complexities, as summarized in Table 2.

2. Wavelet trees

Wavelet trees [9] 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. Rank(B, b, i) counts the number of times bit $b \in \{0, 1\}$ appears in $B[0, i]$, assuming Rank$(B, b, -1) = 0$. The dual operation, Select(B, b, i), returns the position of the i-th occurrence of b, assuming 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 denote 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 $S(v)[i]$ also 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 obtained by choosing $R(v, B(v)[i], i)$ at each node, *i.e.,* at each node we decide to go left or 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 (*[9,11,26]*). *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.*[1] *It solves* RANK, SELECT, *and access to any $S[i]$ in time $O(\log \sigma)$.*

Proof. Grossi et al. [9] proposed a representation using $n \log \sigma + O(\frac{n \log \sigma \log \log n}{\log n}) + O(\sigma \log n)$ 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, Golynski et al. [26] showed how to support binary RANK and SELECT in constant time, while reducing the redundancy of the bitmaps to $O(n \log \log n / \log^2 n)$, which added over the $n \log \sigma$ bits gives $o(n)$ as well. □

2.1. Representation of grids

Consider a set $\mathbf{P}$ of n distinct two-dimensional points (x, y) from a universe $[0, U) \times [0, U)$. We map coordinates to rank space using a standard method [10,5]: 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. 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 $(i, S[i])$. Then we represent S using a wavelet tree.

The space of X and Y corresponds to the bare point data and will not be further mentioned; we will only count the space to store the points in rank space, as usual in the literature. In Appendix A we show how we can represent this mapping into rank space so that, together with a wavelet tree representation, the total space is only $O(n)$ bits over the minimum given by information theory.

The information relative to a point $p_0 = (x_0, y_0)$ is usually tracked from the ROOT and denoted $R(\text{ROOT}, y_0\{..d\}, x_0)$. A pair of points $p_0 = (x_0, y_0)$ and $p_1 = (x_1, y_1)$, where $x_0 \le x_1$ and $y_0 \le y_1$, 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 [11]. 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 $d \le k$ and $V(y_0\{..d'\}) \ne V(y_1\{..d'\})$ 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 $y_0\{..d\} \cdot 1 \ne y_0\{..d + 1\}$, and the nodes $V(y_1\{..d\} \cdot 0)$ such that $y_1\{..d\} \cdot 0 \ne y_1\{..d + 1\}$. These nodes, together with $V(y_0)$ and $V(y_1)$, form the implicit representation of $[y_0, y_1]$, denoted $\text{IMP}(y_0, y_1)$. 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 $Q = [x_0, x_1] \times [y_0, y_1]$, $\text{COUNT}(Q)$, can be computed in time $O(\log n)$ with a structure that requires $n \log n + o(n)$ bits.*

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

This is not the best possible result for this problem (a better result by Bose et al. [4] exists), but it is useful to illustrate how wavelet trees solve range search problems.

3. Geometric queries

In this section we use wavelet trees to solve, within succinct space, two geometric problems of relevance for data analysis.

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

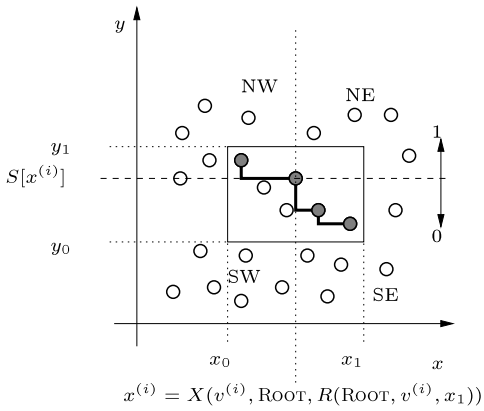

Fig. 1. Dominance on wavelet tree coordinates. The grayed points dominate all the others in the rectangle. We also show the 4 directions.

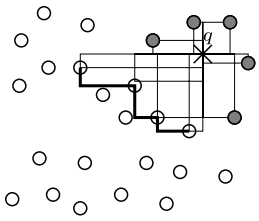

Fig. 2. Rectangle visibility. For SW visibility the problem is the same as dominance. We grayed the points that are visible in the other 3 directions.

3.1. Dominating points

Given points $p_0 = (x_0, y_0)$ and $p_1 = (x_1, y_1)$, we say that p_0 *dominates* p_1 if $x_0 \geq x_1$ and $y_0 \geq y_1$. Note that one point can dominate the other even if they coincide in one coordinate. Therefore, for technical convenience, in the reduction described in Section 2.1, points with the same y coordinates must be ranked in Y by increasing x value, and points with the same x coordinates must be ranked in X by increasing y value. A point is *dominant* inside a range if there is no other point in that range that dominates it. In Fig. 1 we define the cardinal points N, S, E, W, SW, etc. We first use wavelet trees to determine dominant points within rectangles.

Theorem 3. *Given n two-dimensional points, the d dominating points inside a rectangle $Q = [x_0, x_1] \times [y_0, y_1]$ can be obtained (in NW to SE order) in time $O((d+1)\log n)$, with a data structure using $n \log n + o(n)$ bits.*

Proof. Let $v \in \text{IMP}(y_0, y_1)$ be nodes in the implicit representation of $[y_0, y_1]$. We perform depth-first searches (DFS) rooted at each $v \in \text{IMP}(y_0, y_1)$, starting from $V(y_1)$ and continuing sequentially to the left until $V(y_0)$. Each such DFS is computed by first visiting the 1-child and then the 0-child. As a result, we will find the points in N to S order.

We first describe a DFS that reports all the nodes, and then restrict it to the dominant ones. Each visited node v' tracks the interval $(R(\text{ROOT}, v', x_0 - 1), R(\text{ROOT}, v', x_1)]$. If the interval is empty we skip the subtree below v'. As the grid contains only one point per row, for leaves $v' = v^{(i)}$ the intervals $(R(\text{ROOT}, v^{(i)}, x_0 - 1), R(\text{ROOT}, v^{(i)}, x_1)]$ contain at most one value, corresponding to a point $p^{(i)} \in \mathbf{P} \cap Q$. Then $x^{(i)} = Z(v^{(i)}, \text{ROOT}, R(\text{ROOT}, v^{(i)}, x_1))$ and $p^{(i)} = (x^{(i)}, S[x^{(i)}])$. Reporting k points in this way takes $O((k+1)\log n)$ time.

By restricting the intervals associated with the nodes we obtain only dominant points. In general, let v' be the current node, that is either in $\text{IMP}(y_0, y_1)$ or it is a descendant of a node in $\text{IMP}(y_0, y_1)$, and let $(x^{(i)}, S[x^{(i)}])$ be the last point that was reported. Instead of considering the interval $(R(\text{ROOT}, v, x_0 - 1), R(\text{ROOT}, v, x_1)]$, consider the interval $(R(\text{ROOT}, v, x^{(i)}), R(\text{ROOT}, v, x_1)]$. This is correct as it eliminates points (x, y) with $x < x^{(i)}$, and also $y < S[x^{(i)}]$, given the N to S order in which we deliver the points. Fig. 1 illustrates a point $(x^{(i)}, S[x^{(i)}])$.

As explained, a node with an empty interval is skipped. On the other hand, if the interval is non-empty, it must produce at least one dominant point. Hence the cost of reporting the d dominant points amortizes to $O((d+1)\log n)$. □

3.2. Rectangle visibility

Rectangle visibility is another, closely related, geometric problem. A point $p \in \mathbf{P}$ is *visible* from a point $q = (x_0, y_0)$, not necessarily in $\mathbf{P}$, if the rectangle defined by p and q as diagonally opposite corners does not contain any point of $\mathbf{P}$. Depending on the direction from q to p the visibility is called SW, SE, NW or NE (see Fig. 2). Next we solve visibility as a variant of dominance.

Theorem 4. *The structure of Theorem 3 can compute the d points that are visible from a query point $q = (x_0, y_0)$, in order, in time $O((d+1)\log n)$.*

Proof. Note that SW visibility corresponds precisely to determining the dominant points of the region $[0, x_0]\times[0, y_0]$. Hence we use the procedure in Theorem 3. We now adapt it to the three remaining directions without replicating the structure.

For NE or SE visibility, we change the definition of operation R to $R(v, b, i) = \textsc{Rank}(B(v), b, i-1)+1$, thus $\textsc{Rank}(S, c, i) = R(\textsc{Root}, c, i)+1$ if $S[i] = c$ and $\textsc{Rank}(S, c, i) = R(\textsc{Root}, c, i)$ otherwise. In this case we track the intervals $[R(\textsc{Root}, v', x_0), R(\textsc{Root}, v', x_1+1))$. This $R(\textsc{Root}, v', x_1+1)$ is replaced by $R(\textsc{Root}, v', x^{(i)})$ when restricting the search with the last dominant point.

For NE or NW visibility, the DFS searches first visit the 0-child and then use the resulting points to restrict the search on the visit to the 1-child, moreover they first visit the node $V(y_0)$ and move to the right.

Finally, for NW or SE visibility our point ordering in presence of ties in X or Y may report points with the same x or y coordinate. To avoid this we detect ties in X or Y at the time of reporting, right after determining the pair $p^{(i)} = (x, y) = (x^{(i)}, S[x^{(i)}])$. In the NW (SE) case, we binary search for the last (first) positions x' such that $X[x'] = X[x]$ and y' such that $Y[y'] = Y[y]$. Then we correct $p^{(i)}$ to (x', y) (to (x, y')). The subsequent searches are then limited by x' instead of $x = x^{(i)}$. We also limit subsequent searches in a new way: we skip traversing subsequent subtrees of $\textsc{Imp}(y_0, y_1)$ until the y values are larger (NW) or smaller (SE) than y'. Still the cost per reported point is $O(\log n)$. □

3.3. Dynamism

We can support point insertions and deletions on a fixed $U \times U$ grid. Dynamic variants of the bitmaps stored at each wavelet tree node raise the extra space to $o(\log U)$ per point and multiply the times by $O(\log n/\log\log n)$ [27,28].

Lemma 5. *Given n points on a $U \times U$ grid, there is a structure using $n\log U + o(n\log U)$ bits, answering queries in time $O(t(\log U)\log n/\log\log n)$, where $t(h)$ is the time complexity of the query using static wavelet trees of height h. It handles insertions and deletions in time $O(\log U\log n/\log\log n)$.*

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 [27,28]. We also maintain vector X, but not Y; we use the y-coordinates directly instead since the wavelet tree handles repetitions in y. Having a wavelet tree of depth $\log U$ makes the time $t(\log U)$, whereas using dynamic bitmaps multiplies this time by $O(\log n/\log\log n)$,

Instead of an array, we use for X a B-tree tree with arity $\Theta(\log U)$. Nodes are handled with a standard technique for managing cells of different sizes [29], which wastes just $O(\log^2 U)$ bits in total. As a result, the time for accessing a position of X or for finding the range of elements corresponding to a range of coordinates is $O(\log U)$, which is subsumed by other complexities. The extra space on top of that of the bare coordinates is $O(n+\log^2 U)$ bits. This is $o(n\log U)$ unless $n = o(\log U)$, in which case we can just store the points in plain form and solve all queries sequentially. It is also easy to store differentially encoded coordinates in this B-tree to reduce the space of mapping the universe of X coordinates to the same achieved in Section 2.1.

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 $\log U$ bitmaps. Deletion is analogous. □

As a direct application, dominance and visibility queries can be solved in the dynamic setting in time $O((d+1)\log U\log n/\log\log n)$, while supporting point insertions and deletions in time $O(\log U\log n/\log\log n)$. The only issue is that we now may have several points with the same y-coordinate, which is noted when the interval is of size more than one upon reaching a leaf. In this case, as these points are sorted by increasing x-coordinate, we only report the first one (E) or the last one (W). Ties in the x-coordinate, instead, are handled as in the static case.

4. Range sum, average, and variance

We now consider points with an associated value given by an integer function $w : \mathbf{P} \to [0, W)$. We define the sequence of values $W(v)$ associated to each wavelet tree node v as follows: If $S(v) = p_0, p_1, \ldots, p_{|S(v)|}$, then $W(v) = w(p_0), w(p_1), \ldots, w(p_{|S(v)|})$. We start with a solution to several range sum problems on groups. In our results we will omit the bare $n\lceil\log W\rceil$ bits needed to store the values of the points.

Theorem 6. *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 $O(\min(t\log W, \log n)t\log W\log n)$, with a structure that requires $n\log n(1+1/t)$ bits, for any $t \geq 1$. It can also compute the average and variance of the values, $\textsc{Avg}(Q)$ and $\textsc{Var}(Q)$ respectively.*

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 [30] that 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,[2] $\textsc{Wsum}(v, i) = \textsc{Select}(A(v), 1, i+1) - i$ 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:

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

To obtain the tradeoff related to t, we call $\tau = t \log W$ and store only every τ-th entry in A, that is, we store partial sums only at the end of *blocks* of τ entries of $W(v)$. We lose our ability to compute $\textsc{Wsum}(v, i)$ exactly, but can only compute it for i values that are at the end of blocks, $\textsc{Wsum}(v, \tau \cdot i) = \textsc{Select}(A(v), 1, i+1) - i$. To compute each of the terms in the sum of Eq. (1) we can use $\textsc{Wsum}(v, \tau \cdot \lfloor R(\textsc{Root}, v, x_1)/\tau \rfloor) - \textsc{Wsum}(v, \tau \cdot \lceil R(\textsc{Root}, v, x_0 - 1)/\tau \rceil)$ to find the sum of the part of the range that covers whole blocks. Then we must find out the remaining (at most) $2\tau - 2$ values $w(p_i)$ that lie at both extremes of the range, to complete the sum.

In order to find out those values, we store the vectors $W(v)$ explicitly at all the tree nodes v whose height $h(v)$ is a multiple of τ, including the leaves. If a node $v \in \textsc{imp}(y_0, y_1)$ does not have stored its vector $W(v)$, it can still compute any $w(p_i)$ value by tracking it down for at most τ levels.

As a result, the time to compute a $\textsc{Sum}(Q)$ query is $O(\tau^2 \log n)$, yet it is limited to $O(\tau \log^2 n)$ if $\tau > \log n$, as at worst we have $W(v)$ represented at the leaves. The space for the $A(v)$ vectors is at most $(|S(v)|/\tau)(\log W + O(1))$ bits, which adds up to $n \log n(\log W + O(1))/\tau$ bits. On the other hand, the $W(v)$ vectors add up to $n \log n(\log W)/\tau = n(\log n)/t$ bits. This holds for any t even when we store always the $W(v)$ vectors at the leaves: The space of those $W(v)$ is not counted because we take for free the space needed to represent all the values once, as explained.

The average 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, conceptually, an additional instance of the same data structure, with values $w'(p) = w^2(p)$. Then $\textsc{Var}(Q) = \textsc{Sum}'(Q)/\textsc{Count}(Q) - (\textsc{Sum}(Q)/\textsc{Count}(Q))^2$, where $\textsc{Sum}'$ uses the values w'. Note that in fact we only need to store additional (sampled) bitmaps $A'(v)$ corresponding to the partial sums of vectors $W'(v)$ (these bitmaps may need twice the space of the $A(v)$ bitmaps as they handle values that fit in $2 \log W$ bits). Explicitly stored vectors $W'(v)$ are not necessary as they can be emulated with $W(v)$, and we can also share the same wavelet tree structures and bitmaps. This extra space fits within the same $O(n(\log n)/t)$ bits. □

Appendix B shows how to further reduce the constant hidden in the O notation. This is important because this constant is also associated with the $\lceil \log W \rceil$ bits of the weights, that are being omitted from the analysis: In the case of w' we have $2\lceil \log W \rceil$ bits per point.

Finite groups. 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 $\textsc{Wsum}(v, j)^{-1} \oplus \textsc{Wsum}(v, i)$.

4.1. Dynamism

A dynamic variant is obtained by using the dynamic wavelet trees of Lemma 5, a dynamic partial sums data structure instead of $A(v)$, and a dynamic array for vectors $W(v)$.

Theorem 7. *Given n points on a $U \times U$ grid, with associated values in $[0, W)$, there is a structure that uses $n \log U(1+o(1)+1/t)$ bits, for any $t \geq 1$, that answers the queries in Theorem 6 in $O(\log U \log n(1 + \min(t \log W, \log U)t \log W / \log\log n))$ time, and supports point insertions/deletions, and value updates, in time $O(\log U \log n)$.*

Proof. The algorithms on the wavelet tree bitmaps are carried out verbatim, now on the dynamic data structures of Lemma 5, which add $o(n \log U)$ bits of space overhead and multiply the times by $O(\log U / \log\log n)$. This adds $O(\min(t \log W, \log U)t \log W \log U \log n / \log\log n)$ to the query times.

Dynamic arrays to hold the explicit $W(v)$ vectors can be implemented within $|S(v)| \log W(1 + o(1))$ bits, and provide access, insertions and deletions in $O(\log n / \log\log n)$ time [28, Lemma 1]. This adds $O(t \log W \log n / \log\log n)$ to the query times, which is negligible.

For insertions we must insert the new bits at all the levels as in Lemma 5, which costs $O(\log U \log n / \log\log n)$ time, and also insert the new values in $W(v)$ at $1 + (\log U)/(t \log W)$ levels, which in turn costs time $O((1 + (\log U)/(t \log W)) \log n / \log\log n)$ (this is negligible compared to the cost of updating the bitmaps). Deletions are analogous. To update a value we just delete and reinsert the point.

A structure for dynamic searchable partial sums [31] can be represented in $n \log W + o(n \log W)$ bits to store an array of n values, and supports partial sums, as well as insertions and deletions of values, in time $O(\log n)$. Note that we carry out $O(\log U)$ partial sum operations per query. We also perform $O(\log U)$ updates when points are inserted/deleted. This adds $O(\log U \log n)$ time to both query and update complexities.

Maintaining the *sampled* partial sums $A(v)$ is the most complicated part. Upon insertions and deletions we cannot maintain a fixed block size τ. Rather, we use a technique [32] that ensures that the blocks are of length at most 2τ and

[2] Using constant-time $\textsc{Select}$ structures on their internal bitmap H [30].

two consecutive blocks add up to at least τ. This is sufficient to ensure that our space and time complexities hold. The technique does not split or merge blocks, but it just creates/removes empty blocks, and moves one value to a neighboring block. All those operations are easily carried out with a constant number of operations in the dynamic partial sums data structure.

Finally, we need to mark the positions where the blocks start. We can maintain the sequence of $O(|S(v)|/\tau)$ block lengths using again a partial sums data structure [31], which takes $O((|S(v)|/\tau)\log\tau)$ bits. The starting position of any block is obtained as a partial sum, in time $O(\log n)$, and the updates required when blocks are created or change size are also carried out in time $O(\log n)$. These are all within the same complexities of the partial sum structure for $A(v)$. □

Finite groups and semigroups. The solution applies to finite groups $(G, \oplus, ^{-1}, 0)$. The dynamic structure for partial sums [31] can be easily converted into one that stores the local "sum" $w(p_i) \oplus w(p_{i+1}) \oplus \ldots w(p_j)$ of each subtree containing leaves $p_i, p_{i+1}, \ldots p_j$. The only obstacle in applying it to semigroups is that we cannot move an element from one block to another in constant time, because we have to recalculate the "sum" of a block without the element removed. This takes time $O(\tau)$, so the update time becomes $O(\log U(\log n + t\log W))$.

5. 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 [33]. 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 relative order. So we set up a bitmap $V[1, W]$ where the values occurring in the set are marked. This bitmap can be stored within at most $m\log(W/m) + O(m)$ bits [30], where $m \le \min(n, W)$ is the number of unique values. This representation converts between actual value and relative order in time $O(\log m)$, which will be negligible. This way, many complexities will be expressed in terms of m.

Theorem 8. *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 $O(\min(t\log m, \log n)t\log n)$, with a structure using $n\log n(1 + 1/t)$ bits, for any $t \ge 1$ and $m = \min(n, W)$. The maximum of the point values inside a query rectangle Q can be found within the same time and space bounds.*

Proof. We associate to each node v the one-dimensional data structure [33] corresponding to $W(v)$, which takes $2|W(v)| + o(|W(v)|)$ bits. This adds up to $2n\log n + o(n\log n)$ bits overall. We call $\textsc{Wmin}(v, i, j) = \arg\min_{i\le s\le j} W(v)[s]$ 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). 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 $W(v)[s]$ of different nodes v. By storing the $W(v)$ vectors of Theorem 6 (in the range $[1, m]$) every $\tau = t\log m$ levels, the time is just $O(\min(\tau, \log n)\log n)$ because we have to track down just one point for each $v \in \textsc{imp}(y_0, y_1)$. The space is $3n\log n + (n\log n\log m)/\tau = 3n\log n + (n\log n)/t$ bits. The second term holds for any t even when we always store $n\log m$ bits at the leaves, because adding these to the $m\log(W/m) + O(m)$ bits used for V, we have the $n\lceil\log W\rceil$ bits corresponding to storing the bare values and that are not accounted for in our space complexities.

To reduce the space further, we split $W(v)$ into blocks of length r and create a sequence $W'(v)$, of length $|S(v)|/r$, where we take the minimum of each block, $W'(v)[i] = \min\{W(v)[(i-1)\cdot r + 1], \ldots, W(v)[i\cdot r]\}$. The one-dimensional data structures are built over $W'(v)$, not $W(v)$, and thus the overall space for these is $O((n/r)\log n)$ bits. In exchange, to complete the query we need to find the r values covered by the block where the minimum was found, plus up to $2r - 2$ values in the extremes of the range that are not covered by full blocks. The time is thus $O(r\min(\tau, \log n)\log n)$. By setting $r = t$ we obtain the result.

For $\textsc{Max}(Q)$ we use analogous data structures. □

Top-k queries in succinct space. We can now solve the top-k query of Rahul et al. [16] by iterating over Theorem 8. Let us set $r = 1$. Once we identify that the overall minimum is some $W(v)[s]$ from the range $W(v)[i, j]$, we can find the second minimum among the other candidate ranges plus the ranges $W(v)[i, s-1]$ and $W(v)[s+1, j]$. As this is repeated k times, we pay time $O(\tau(k + \log n))$ to find all the minima. 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 $O(\tau\log n + k(\tau + \log k))$ by using a priority queue with constant insertion time [34]. Using $\tau = t\log m$ for any $t = \omega(1)$ we obtain time $O(t\log m\log n + kt\log m\log k)$ and $n\log n + o(n\log n)$ bits of space. The best current linear-space solution [24] achieves better time and linear space, but the constant multiplying the linear space is far from 1.

5.1. Dynamism

We can directly apply the result on semigroups given in Section 4.1. Note that, while in the static scenario we achieve a better result than for sums, in the dynamic case the result is slightly worse.

Theorem 9. *Given n points on an $U \times U$ grid, with associated values in $[0, W)$, there is a structure using $n \log U(1 + o(1) + 1/t)$ bits, for any $t \geq 1$, that answers the queries in Theorem 8 in $O(\log U \log n(1 + \min(t \log W, \log U)t \log W / \log\log n))$ time, and supports point insertions and deletions, and value updates, in $O(\log U(\log n + t \log W))$ time.*

6. Range majority for fixed α

In this section we describe a data structure that answers α-majority queries for the case where α is fixed at construction time. Again, we enrich the wavelet tree with additional information that is sparsified. We obtain the following result.

Theorem 10. *Given n two-dimensional points with associated values in $[0, W)$ and a fixed value $0 < \alpha < 1$, all the α-majorities inside a query rectangle $Q = [x_0, x_1] \times [y_0, y_1]$, $\textsc{Majority}(\alpha, Q)$, can be found in time $O(t \log m \log^2 n)$, with a structure using $n((2 + 1/t) \log n + \log m)$ bits, for any $t \geq 1$ and $m = \min(n, W)$.*

Proof. We say that a set of values C is a set of *α-candidates* for $S' \subset S$ if each α-majority value w of S' belongs to C. In every wavelet tree node v we store an auxiliary data structure $A(v)$ that corresponds to elements of $W(v)$. The data structure $A(v)$ enables us to find the set of α-candidates for any range $[r_1 \cdot s, r_2 \cdot s]$ in $W(v)$, for a parameter $s = t \log m$. We implement $A(v)$ as a balanced binary range tree $T(v)$ on $W(v)$. Every leaf of $T(v)$ corresponds to an interval of $W(v)$ of size s/α. The range of an internal node w of T is the union of ranges associated with its children. In every such node w, we store all α-majority values for the range of w (these are at most $1/\alpha$ values). The space required by $T(v)$ is $(2|W(v)|/(s/\alpha))(1/\alpha) \log m = O(|W(v)|/t)$ bits, which added over all the wavelet tree nodes v sums to $(n \log n)/t$ bits.

Given an interval $[r_1 \cdot t, r_2 \cdot t]$, we can represent it as a union of $O(\log n)$ ranges for nodes $w_i \in T(v)$. If a value is an α-majority value for $[r_1 \cdot t, r_2 \cdot t]$, then it is an α-majority value for at least one w_i. Hence, a candidate set for $[r_1 \cdot t, r_2 \cdot t]$ is the union of values stored in w_i. The candidate set contains $O((1/\alpha) \log n)$ values and can be found in $O((1/\alpha) \log n)$ time.

Moreover, for every value c, we store the grid $G(c)$ of Lemma 2, which enables us to find the total number of elements with value w in any range $Q = [x_0, x_1] \times [y_0, x_1]$. Each $G(c)$, however, needs to have the coordinates mapped to the rows and columns that contain elements with value c. We store sequences X_c and Y_c giving the value identifiers of the points sorted by x- and y-coordinates, respectively. By representing them as wavelet trees, they take $2n \log m + o(n)$ bits of space and map in constant time any range $[x_0, x_1]$ or $[y_0, y_1]$ using *rank* operations on the sequences, in $O(\log m)$ time using the wavelet trees. Then the local grids, which overall occupy other $\sum_{c \in [1,m]} n_c \log n_c + o(n_c) \leq n \log(n/m) + o(n)$ bits, complete the range counting query in time $O(\log n)$. So the total space of these range counting structures is $n \log n + n \log m + o(n)$.

To solve an α-majority query in a range $Q = [x_1, x_2] \times [y_1, y_2]$, we visit each node $v \in \textsc{imp}(y_0, y_1)$. We identify the longest interval $[r_1 \cdot s, r_2 \cdot s] \subseteq [R(\textsc{Root}, v, x_0), R(\textsc{Root}, v, x_1)]$. Using $A(v)$ the candidate values in $[r_1 \cdot s, r_2 \cdot s]$ can be found in time $O((1/\alpha) \log n)$. Then we obtain the values of the elements in $[R(\textsc{Root}, v, x_0), r_1 \cdot s)$ and $(r_2 \cdot s, R(\textsc{Root}, v, x_1)]$, in time $O(s \log n)$ by traversing the wavelet tree. Added over all the $v \in \textsc{imp}(y_0, y_1)$, the cost to find the $(1/\alpha + s) \log n$ candidates is $O((1/\alpha + s \log n) \log n)$. Then their frequencies in Q are counted using the grids $G(c)$ in time $O((1/\alpha + s) \log^2 n)$, and the α-majorities are finally identified.

Thus the overall time is $O(t \log m \log^2 n)$. The space is $n(2 \log n + \log m + (\log n)/t)$, higher than for the previous problems but less than the structures to come. □

A slightly better (but messier) time complexity can be obtained by using the counting structure of Bose et al. [4] instead of that of Lemma 2, storing value identifiers every s tree levels, $O(t \log m \log n(\min(t \log m, \log n) + \log n / \log\log n))$. The space increases by $o(n \log(n/m))$. On the other hand, by using $s = t = 1$ we increase the space to $O(n \log n)$ integers and reduce the query time to $O(\log^2 n)$.

7. 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 $[0, m)$ 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 $\lfloor w(p)/2^{h(v)} \rfloor = L(v\{..\lceil \log m \rceil - h(v)\})$. Note that each leaf c stores the points p with value $w(p) = c$.

Theorem 11. *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 $O(\ell \log n \log_\ell m)$, with a structure using $n \log n \log_\ell m + O(n \log m)$ bits, for any $\ell \in [2, m]$ and $m = \min(n, W)$.*

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 $\text{RANK}(X(v), b, x_0)$ and x_1 to $\text{RANK}(X(v), b, x_1 + 1) - 1$, and analogously for y_0 and y_1 with $Y(v)$.

The bitmaps $X(v)$ and $Y(v)$ add up to $O(n \log m)$ 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 $\lceil \log m \rceil - 1$ 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 $\text{COUNT}(Q)$, but this gives sufficient information to directly descend $\log \ell$ levels. Thus total time adds up to $O(\ell \log n \log_\ell m)$. □

Again, by replacing our structure of Lemma 2 by Bose et al.'s counting structure [4], the time drops to $O(\ell \log n \log_\ell m / \log \log n)$ when using $n \log n \log_\ell m(1 + o(1)) + O(n \log m)$ bits of space.

The basic wavelet tree structure 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 $O(\ell \log n \log_\ell m)$ or $O(\ell \log n \log_\ell m / \log \log n)$. This is another useful operation for data analysis, and can be obtained with the formula $\sum_{v \in \text{IMP}(w_0, w_1)} \text{COUNT}(Q)$.

As a curiosity, we have tried, just as done in Sections 4 and 5, to build a wavelet tree on the y-coordinates and use a one-dimensional data structure. We used the optimal linear-space structure of Brodal and Jørgensen [20]. However, the result is not competitive with the one we have achieved by building a wavelet tree on the domain of point values.

8. Range majority for variable α

We can solve this problem, where α is specified at query time, with the same structure used for Theorem 11.

Theorem 12. *The structures of Theorem 11 can compute all the α-majorities of the point values inside Q, $\text{MAJORITY}(\alpha, Q)$, in time $O(\frac{1}{\alpha} \ell \log n \log_\ell m)$, 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 $\alpha \cdot \text{COUNT}(Q)$, then c is the answer, else there is no α-majority. For $\alpha < \frac{1}{2}$, we solve the query by probing all the $(i \cdot \alpha)\text{COUNT}(Q)$-th elements in Q. □

Once again, we attempted to build a wavelet tree on y-coordinates, using the one-dimensional structure of Durocher et al. [22] at each level, but we obtain inferior results.

Culpepper et al. [35] show how to find the mode, and in general the k most repeated values inside Q, using successively more refined QUANTILE queries. Let the k-th most repeated value occur $\alpha \cdot \text{COUNT}(Q)$ times in Q, then we require at most $4/\alpha$ quantile queries [35]. The same result can be obtained by probing successive values $\alpha = 1/2^i$ with $\text{MAJORITY}(\alpha)$ queries.

9. 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) value larger (smaller) than, or equal to, w in Q. We also have an efficient solution using our wavelet trees on grids.

Theorem 13. *The structures of Theorem 11 can compute the successor and predecessor of a value w within the values of the points inside Q, $\text{SUCC}(w, Q)$ and $\text{PRED}(w, Q)$, in time $O(\ell \log n \log_\ell m)$.*

Proof. We consider the nodes $v \in \text{IMP}(w, +\infty)$ from left to right, tracking rectangle Q in the process. The condition for continuing the search below a node v that is in $\text{IMP}(w, +\infty)$, or is a descendant of one such node, is that $\text{COUNT}(Q) > 0$ on $G(v)$. $\text{SUCC}(w, Q)$ is the value associated with the first leaf found by this process. Likewise, $\text{PRED}(w, Q)$ is computed by searching $\text{IMP}(-\infty, w)$ 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 Theorem 11, the total time amortizes to $O(\ell \log n \log_\ell m)$. □

Once again, storing one-dimensional data structures [36,37] on a y-coordinate-based wavelet tree does not yield competitive results.

10. Dynamism

Our dynamic wavelet tree of Lemma 5 supports range counting and point insertions/deletions on a fixed grid in time $O(\log U \log n / \log \log n)$ (other tradeoffs exist [2]). If we likewise assume that our grid is fixed in Theorems 11–13, we can also support point insertions and deletions (and thus changing the value of a point).

Theorem 14. *Given n points on a $U \times U$ grid, with associated values in $[0, W)$, there is a structure using $n \log U \log_\ell W(1+o(1))$ bits, for any $\ell \in [2, W]$, that answers the queries* QUANTILE, SUCC *and* PRED *in time $O(\ell \log U \log n \log_\ell W / \log \log n)$, and the* $\text{MAJORITY}(\alpha)$ *operations in time $O(\frac{1}{\alpha} \ell \log U \log n \log_\ell W / \log \log n)$. It supports point insertions and deletions, and value updates, in time $O(\log U \log n \log_\ell W / \log \log n)$.*

Proof. We use the data structure of Theorems 11–13, 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 5, on global y-coordinates $[0, U)$. We maintain the global array X of Lemma 5 plus the vectors $X(v)$ of Theorem 11, the latter using dynamic bitmaps [27,28]. 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 $X(v)$. □

11. Conclusions

We have demonstrated how wavelet trees [9] can be used for solving a wide range of two-dimensional queries that are useful for various data analysis activities. Wavelet trees have the virtue of using little space. By enriching them with further sparsified data, we support various complex queries in polylogarithmic time and linear space, sometimes even succinct. Other more complicated queries require slightly superlinear space.

We believe this work just opens the door to the possible applications to data analysis, and that many other 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 [38] and even in two dimensions [25], but the results are far from satisfactory.

Another interesting open problem is how to support dynamism while retaining time complexities logarithmic in the number of points and not in the grid size. This is related to the problem of dynamic wavelet trees, in particular supporting insertion and deletion of y-coordinates (on which they build the partition). Dynamic wavelet trees would also solve many problems in other areas.

Finally, a natural question is which are the lower bounds that relate the achievable space and time complexities for the data analysis queries we have considered. These are well known for the more typical counting and reporting queries, but not for these less understood ones.

Acknowledgments

The first and second authors were supported in part by Millennium Nucleus Information and Coordination in Networks ICM/FIC P10-024F, Mideplan, Chile. The third author was supported by FCT by national funds through FCT Fundação para a Ciência e a Tecnologia, under projects PEst-OE/EEI/LA0021/2011, TAGS PTDC/EIA-EIA/112283/2009 and HELIX PTDC/EEA-ELC/113999/2009.

Appendix A. Optimal-space representation of grids

We analyze the representation described in Section 2.1, showing how it can be made near-optimal in the information-theoretic sense. Recall that our representation of a set of points of $[0, U)^2$ consists in storing two sorted arrays X and Y, which reduce the $[0, U)$ values to $[0, n)$. The points in the $[0, n) \times [0, n)$ grid have exactly one point per row and one per column.

An optimal-space representation of the above data uses the data structure of Okanohara and Sadakane [30] for mapping the sorted X coordinates (where point $X(i)$ is represented as a bit set at position $i+X(i)$ in a bitmap of length $n+U$), a similar structure for the Y coordinates, and a wavelet tree for the grid of mapped points. The former occupies $n \log \frac{n+U}{n} + O(n) = \log \binom{U+n}{n} + O(n)$ bits of space, gives constant-time access to the real coordinate of any point $X(i) = select_1(i) - i$, and takes $O(\log \frac{U+n}{n})$ time to map any value x to rank space at query time; and similarly for Y. The wavelet tree requires $n \log n + o(n) = \log n! + O(n)$ bits. Overall, if we ignore the $O(n)$-bit redundancies, the total space is $\log \left(n! \binom{U+n}{n}^2 \right)$ bits.

Hence our representation can be in any of $n!\binom{U+n}{n}^2$ configurations. Note that we can represent repeated points, which is useful in some cases, especially when they can have associated values. We show now that our number of configurations is not much more than the number of possible configurations of **P** even if repeated points are forbidden, *i.e.*, n distinct points from $[0, U)^2$ can be in $\binom{U^2}{n} \leq \binom{U+n}{n}^2 n!$ configurations. The difference is not so large because $\binom{U+n}{n}^2 n! \leq \binom{U^2}{n} c^n$, for any $c \geq 4$. In terms of bits this means that $2 \log \binom{U+n}{n} + \log n! \leq \log \binom{U^2}{n} + n \log c$ and therefore our representation is at most $O(n)$ bits larger than an optimal representation, aside from the $O(n)$ bits we are already wasting with respect to $\log \left(n! \binom{U+n}{n}^2 \right)$.

To see this, notice that $\binom{U+n}{n}^2 n! = ((U+n)!/(n!U!))^2 n! = ((U+n)!/U!)^2/n! = (\prod_{i=0}^{n-1}(U+n-i)^2)/n!$. For sufficiently large c, this is $\leq (\prod_{i=0}^{n-1} c(U^2-i))/n! = c^n U^2!/((U^2-n)!n!) = \binom{U^2}{n} c^n$.

We need $c \geq (U+n-i)^2/(U^2-i)$ for any $0 \leq i < n$. We next show that, if $n \leq U$, then $(U+n)^2/U^2 \geq (U+n-i)^2/(U^2-i)$, and thus it is enough to choose $c \geq (U+n)^2/U^2$. Simple algebra shows that the condition is equivalent to $i \leq 2(U+n) - (1 + (n/U))^2$. Since we assume for now that $n/U \leq 1$, the inequality is satisfied if $i \leq 2(U+n) - 4$. Since $i < n$, the inequality always holds (as $U, n \geq 1$). Thus it is sufficient that $c \geq (U+n)^2/U^2$, which is no larger than 4 if $n \leq U$.

Let us now consider the case $n > U$. Our analysis still holds, up large enough n. Since $i \leq n-1$, it is sufficient that $n-1 \leq 2(U+n)-(1+(n/U))^2$. Simple algebra shows this is equivalent to the cubic inequality $2U^3+nU^2-2nU-n^2 \geq 0$. As a function of U this function has three roots, the only positive one at $U=\sqrt{n}$. Therefore it is positive for $U \geq \sqrt{n}$, *i.e.*, $n \leq U^2$, which covers all the possible values of n.

Appendix B. Reducing space for variance

We now discuss how to bring the $2\lceil \log W \rceil$ space factor associated to storing weights $w'(p) = w(p)^2$ closer to $\lceil \log V \rceil$, where V is the overall variance.

Instead of storing $w'(p)$, we can store $w''(p) = (w(p) - \lceil T/n \rceil)^2$, where T/n is the average of all the points. To obtain $\textsc{Var}(Q)$ from the sum of the w'' in Q notice that $(w(p)-(T_Q/q))^2 = (w(p)-\lceil T/n \rceil)^2 - 2(w(p)-\lceil T/n \rceil)(\lceil T/n \rceil - (T_Q/q)) + ((T/n)-(T_Q/q))^2$, where $T_Q = \textsc{Sum}(Q)$ and $q = \textsc{Count}(Q)$. This formula makes use of the stored $w''(p)$ values, as well as queries $\textsc{Sum}(Q)$ and $\textsc{Count}(Q)$. Note that rounding is used to keep the values as integers, hence limiting the number of bits necessary in its representation.

To avoid numeric instability and wasted space, it is better that T/n is close to T_q/q. This simultaneously yields smaller $(w(p) - \lceil T/n \rceil)^2$ values and reduces the $(\lceil T/n \rceil - (T_Q/q))$ factor in the subtraction. To ensure this we may (logically) partition the space and use the local average, instead of the global one. Each level of the wavelet tree partitions the space into horizontal non-overlapping bands, of the form $[j \cdot n/2^k, (j+1) \cdot n/2^k)$, for some k. At every level we use the average of the band in question. This allows us to compute variances for rectangles whose y coordinates are aligned with the bands, while the x coordinates are not restricted. For a general rectangle Q we decompose it into band-aligned rectangles, just as with any other query on the wavelet tree (recall Eq. (1)). Alternatively, we can use a variance update formula [39,40] that is stable and further reduces the instability of the first calculation. We rewrite the update formula of Chan et al. [39] in terms of sets, since originally it was formulated in one dimension.

Lemma 15 (*[39, Eq. 2.1]*)**.** *Given two disjoint sets A and B, which have values $w(p)$ associated with element p, where $T_A = \textsc{Sum}(A)$, $T_B = \textsc{Sum}(B)$, $m = \textsc{Count}(A)$, $n = \textsc{Count}(B)$, $S_A = \sum_{p \in A}(w(p) - T_A/m)^2$ and $S_B = \sum_{p \in B}(w(p) - T_B/n)^2$, the following equalities hold:*

$$T_{A \bigcup B} = T_A + T_B \tag{B.1}$$

$$S_{A \bigcup B} = S_A + S_B + \frac{m}{n(m+n)}\left(\frac{n}{m}T_A - T_B\right)^2. \tag{B.2}$$

Notice that $\textsc{Var}(A) = S_A$.

References

[1] M. Berg, O. Cheong, M. Kreveld, M. Overmars, Orthogonal range searching: querying a database, in: Computational Geometry, Springer, 2008, pp. 95–120.
[2] Y. Nekrich, Orthogonal range searching in linear and almost-linear space, Computational Geometry Theory and Applications 42 (2009) 342–351.
[3] M. Pătraşcu, Lower bounds for 2-dimensional range counting, in: Proc. 39th Annual ACM Symposium on Theory of Computing, STOC, 2007, pp. 40–46.
[4] P. Bose, M. He, A. Maheshwari, P. Morin, Succinct orthogonal range search structures on a grid with applications to text indexing, in: Proc. 11th International Symposium on Algorithms and Data Structures, WADS, in: LNCS 5664, 2009, pp. 98–109.
[5] S. Alstrup, G. Brodal, T. Rauhe, New data structures for orthogonal range searching, in: Proc. 41st Annual Symposium on Foundations of Computer Science, FOCS, 2000, pp. 198–207.
[6] M. Patrascu, M. Thorup, Time-space trade-offs for predecessor search, in: Proc. 38th ACM Symposium on Theory of Computing, STOC, 2006, pp. 232–240.
[7] T.M. Chan, K. Larsen, M. Pătraşcu, Orthogonal range searching on the RAM, revisited, in: Proc. 27th Annual Symposium on Computational Geometry, SoCG, 2011, pp. 1–10.
[8] G. Brodal, K. Tsakalidis, Dynamic planar range maxima queries, in: Proc. 38th International Colloquium on Automata, Languages and Programming, ICALP, 2011, pp. 256–267.
[9] R. Grossi, A. Gupta, J. Vitter, High-order entropy-compressed text indexes, in: 14th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA, 2003, pp. 841–850.
[10] B. Chazelle, A functional approach to data structures and its use in multidimensional searching, SIAM Journal on Computing 17 (1988) 427–462.
[11] V. Mäkinen, G. Navarro, Rank and select revisited and extended, Theoretical Computer Science 387 (2007) 332–347.
[12] T. Gagie, S. Puglisi, A. Turpin, Range quantile queries: another virtue of wavelet trees, in: Proc. 16th International Symposium on String Processing and Information Retrieval, SPIRE, 2009, pp. 1–6.
[13] N. Brisaboa, M. Luaces, G. Navarro, D. Seco, A fun application of compact data structures to indexing geographic data, in: Proc. 5th International Conference on Fun with Algorithms, FUN, in: LNCS 6099, 2010, pp. 77–88.
[14] J. Barbay, F. Claude, G. Navarro, Compact rich-functional binary relation representations, in: Proc. 9th Latin American Symposium on Theoretical Informaticsm, LATIN, in: LNCS 6034, 2010, pp. 170–183.
[15] T. Gagie, G. Navarro, S. Puglisi, New algorithms on wavelet trees and applications to information retrieval, Theoretical Computer Science 426–427 (2012) 25–41.
[16] S. Rahul, P. Gupta, R. Janardan, K. Rajan, Efficient top-k queries for orthogonal ranges, in: Proc. 5th International Workshop on Algorithms and Computation, WALCOM, in: LNCS 6552, 2011, pp. 110–121.
[17] D. Willard, New data structures for orthogonal range queries, SIAM Journal on Computing 14 (1985).
[18] A. Amir, J. Fischer, M. Lewenstein, Two-dimensional range minimum queries, in: Proc. 18th Annual Symposium on Combinatorial Pattern Matching, CPM, in: LNCS 4580, 2007, pp. 286–294.
[19] G. Brodal, P. Davoodi, S. Rao, On space efficient two dimensional range minimum data structures, Algorithmica (2011) http://dx.doi.org/10.1007/s00453-011-9499-0.

[20] G. Brodal, A. Jørgensen, Data structures for range median queries, in: Proc. 20th International Symposium on Algorithms and Computation, ISAAC, in: LNCS 5878, 2009, pp. 822–831.
[21] M. Karpinski, Y. Nekrich, Searching for frequent colors in rectangles, in: Proc. 20th Canadian Conference on Computational Geometry, CCCG, 2008.
[22] S. Durocher, M. He, I. Munro, P. Nicholson, M. Skala, Range majority in constant time and linear space, in: Proc. 38th International Colloquium on Automata, Languages and Programming, ICALP, pp. 244–255.
[23] T. Gagie, M. He, J. Munro, P. Nicholson, Finding frequent elements in compressed 2d arrays and strings, in: Proc. 18th International Symposium on String Processing and Information Retrieval, SPIRE, in: LNCS 7024, pp. 295–300.
[24] G. Navarro, Y. Nekrich, Top-k document retrieval in optimal time and linear space, in: Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA, 2012 (in press).
[25] S. Durocher, J. Morrison, Linear-space data structures for range mode query in arrays, CoRR 1101.4068, 2011.
[26] A. Golynski, R. Grossi, A. Gupta, R. Raman, S.S. Rao, On the size of succinct indices, in: Proc. 15th Annual European Symposium on Algorithms, ESA, in: LNCS 4698, pp. 371–382.
[27] M. He, I. Munro, Succinct representations of dynamic strings, in: Proc. 17th International Symposium on String Processing and Information Retrieval, SPIRE, 2010, pp. 334–346.
[28] G. Navarro, K. Sadakane, Fully-functional static and dynamic succinct trees, CoRR 0905.0768v5, 2010.
[29] J.I. Munro, An implicit data structure supporting insertion, deletion, and search in $O(\log n)$ time, Journal of Computer Sysstems Science 33 (1986) 66–74.
[30] D. Okanohara, K. Sadakane, Practical entropy-compressed rank/select dictionary, in: Proc. Workshop on Algorithm Engineering and Experiments, ALENEX, 2007.
[31] V. Mäkinen, G. Navarro, Dynamic entropy-compressed sequences and full-text indexes, ACM Transactions on Algorithms 4 (2008) Art. 32.
[32] R. González, G. Navarro, Rank/select on dynamic compressed sequences and applications, Theoretical Computer Science 410 (2008) 4414–4422.
[33] J. Fischer, Optimal succinctness for range minimum queries, in: Proc. 9th Latin American Symposium on Theoretical Informatics, LATIN, in: LNCS 6034, 2010, pp. 158–169.
[34] S. Carlsson, J.I. Munro, P.V. Poblete, An implicit binomial queue with constant insertion time, in: Proc. 1st Scandinavian Workshop on Algorithmic Theory, SWAT, 1988, pp. 1–13.
[35] S. Culpepper, G. Navarro, S. Puglisi, A. Turpin, Top-k ranked document search in general text databases, in: Proc. 18th Annual European Symposium on Algorithms, ESA, in: LNCS 6347, pp. 194–205 (part II).
[36] M. Crochemore, C.S. Iliopoulos, M. Kubica, M. Rahman, T. Walen, Improved algorithms for the range next value problem and applications, in: Proc. 25th Symposium on Theoretical Aspects of Computer Science, STACS, pp. 205–216.
[37] V. Mäkinen, G. Navarro, E. Ukkonen, Transposition invariant string matching, Journal of Algorithms 56 (2005) 124–153.
[38] T. Gagie, G. Navarro, S. Puglisi, Colored range queries and document retrieval, in: Proc. 17th International Symposium on String Processing and Information Retrieval, SPIRE, 2010, pp. 67–81.
[39] T. Chan, G. Golub, R. LeVeque, Updating Formulae and a Pairwise Algorithm for Computing Sample Variances, Technical Report STAN-CS-79-773, Dept. of CS, Stanford Univ., 1979.
[40] B.P. Welford, Note on a method for calculating corrected sums of squares and products, Technometrics 4 (1962) 419–420.