Entropy-Bounded Representation of Point Grids ☆

Arash Farzan[a], Travis Gagie[b], Gonzalo Navarro[c,1]

[a] *Max-Planck-Institut für Informatik, Germany.* afarzan@mpi-inf.mpg.de
[b] *Dept. of Computer Science, Aalto University, Finland.* travis.gagie@aalto.fi
[c] *Dept. of Computer Science, University of Chile, Chile.* gnavarro@dcc.uchile.cl

Abstract

We give the first *fully compressed* representation of a set of m points on an $n \times n$ grid, taking $H + o(H)$ bits of space, where $H = \lg \binom{n^2}{m}$ is the entropy of the set. This representation supports range counting, range reporting, and point selection queries, with complexities that go from $\mathcal{O}(1)$ to $\mathcal{O}\big(\lg^2 n / \lg\lg n\big)$ per answer as the entropy of the grid decreases. Operating within entropy-bounded space, as well as relating time complexity with entropy, opens a new line of research on an otherwise well-studied area.

Keywords: Compressed data structures, geometric grids, range queries.

1. Introduction

A point grid is a basic structure underlying the representation of two-dimensional point sets, graphics, spatial databases, geographic data, binary relations, graphs, images, and so on. It has been intensively studied from a computational geometry viewpoint, where most of the focus has been on two basic primitives: (orthogonal) range counting (how many points are there in this rectangle?), and (orthogonal) range reporting (list the points falling within this rectangle). More sophisticated setups include more complex queries like finding dominant points, querying shapes more general than rectangles, computing aggregates on values associated to points, etc. [1]. In most cases these more complex queries build on the two basic primitives

☆ An early version of this article appeared in *Proc. ISAAC 2010* [13].

[1] Partially funded by Millennium Nucleus Information and Coordination in Networks ICM/FIC P10-024F, Chile.

mentioned, plus a third one we call point selection (which is the kth point in this range?).

Consider an $n \times n$ grid containing m points, and the RAM computation model with word size $w = \Theta(\lg n)$. Currently the best results related to the focus of this paper are as follows. Range counting can be done in time $\mathcal{O}\left(\frac{\lg m}{\lg \lg m}\right)$ and linear space, that is, $\mathcal{O}(m)$ words [23]. That counting time cannot be improved within $\mathcal{O}(m \operatorname{polylog}(m))$ words space [28]. Range reporting can be done in time $\mathcal{O}(\lg \lg m + k)$, where k is the number of points reported, using $\mathcal{O}(m \lg^{\epsilon} m)$ words for any constant $\epsilon > 0$ [2]. This time, again, is optimal within $\mathcal{O}(m \operatorname{polylog}(m))$ words space [7]. It rises to $\mathcal{O}((k+1) \lg \lg m)$ if the space is reduced to $\mathcal{O}(m \lg \lg m)$ words, and it reaches $\mathcal{O}((k+1) \lg^{\epsilon} m)$ if the space is $\mathcal{O}(m)$ words [7]. There are also some bounds that may be relevant when many points are to be reported, as the cost per reported point decreases with k: $\mathcal{O}(\lg m + k \lg \lg(4m/k))$ time using $\mathcal{O}(m \lg \lg m)$ words, and $\mathcal{O}(\lg m + k \lg^{\epsilon}(2m/k))$ time using $\mathcal{O}(m)$ words [9]. Some of these results have been matched even in the dynamic scenario [25].

Many of the application areas for this problem handle huge volumes of information, and in those cases superlinear-space structures are impractical. Even the linear-space structures (i.e., $\mathcal{O}(m)$ words, or $\mathcal{O}(m \lg n)$ bits) might be excessively large. On top of the coordinates of the points, they add several auxiliary structures that add to the constant factor multiplying the $\mathcal{O}(m \lg n)$ term. When space is a concern, one can aim not only at using linear space, but at *succinctness*, that is, using $m \lg n(1 + o(1))$ bits of space.

A few succinct data structures exist. Bose et al. [6] presented a structure using $m \lg m + o(m \lg m)$ bits to store $m = n$ points in an $n \times n$ grid, answering range counting queries in time $\mathcal{O}\left(\frac{\lg m}{\lg \lg m}\right)$ and reporting in time $\mathcal{O}\left((k+1)\frac{\lg m}{\lg \lg m}\right)$. Another proposal approaching succinctness is by Barbay et al. [3], which uses $m \lg n + o(m) \lg n + \mathcal{O}(m+n)$ bits. Within this space they solve many interesting range queries including counting, reporting, and point selection, in $\mathcal{O}(\lg n)$ and even $\mathcal{O}(\lg n / \lg \lg n)$ time per answer.

Even a succinct space like $m \lg n$ bits is not the best possible for all values of m and n. A (worst-case) lower bound on the number of bits needed to represent a grid is the logarithm of the number of possible grids, called the

"entropy":

$$\begin{aligned} H &= \lg \binom{n^2}{m} \\ &= m \lg \frac{n^2}{m} + (n^2 - m) \lg \frac{n^2}{n^2 - m} + \mathcal{O}(\lg n) \qquad (1) \\ &= m \lg \frac{n^2}{m} + (n^2 - m) \lg \left(1 + \frac{m}{n^2 - m}\right) + \mathcal{O}(\lg n) \\ &= m \lg \frac{n^2}{m} + \mathcal{O}(m + \lg n)\,. \end{aligned}$$

In this paper we push further in the direction of storing the grid data within its entropy bound. Most notably, we achieve a *fully compressed* representation taking $H + o(H)$ bits of space. While the worst-case time we achieve for the operations is $\mathcal{O}\big(\lg^2 n / \lg\lg n\big)$, we obtain for most values of m and n time complexities of the form $\mathcal{O}\Big(\lg \frac{n^2}{m} / \lg\lg n\Big)$, which improves as the grid becomes denser (i.e., as the entropy increases), reaching even constant time when $m = \Omega(n^2/\mathrm{polylog}(n))$. See Table 1 for the precise details. We are not aware of previous works relating the entropy with the time complexities of the operations, only between the space and the time in terms of m.

The paper is organized as follows. Section 2 gives basic concepts on bitmaps and point grids, defines the problems we address, proves some technical results needed later and summarizes the results we achieve. Section 3 gives two simple solutions obtained by plugging in existing results. While they get close to reaching entropy space, $H + \mathcal{O}(m)$ bits, their times are insensitive to the entropy, typically $\mathcal{O}(\lg m / \lg\lg m)$. Section 4 describes a representation taking $H + o(n^2)$ bits and achieving constant time for range counting and reporting. Such a redundancy is $o(H)$ when the matrix is dense enough (but not extremely dense). Section 5 achieves $H + o(H) + \mathcal{O}(m \lg\lg\lg m)$ bits, in exchange for higher query times, namely the entropy-sensitive $\mathcal{O}\Big(\lg \frac{n^2}{m} / \lg\lg n\Big)$. This redundancy is $o(H)$ when the matrix is sparse enough. Finally, Section 6 combines the previous results to finally reach the $H + o(H)$ bits in all cases, yet the times become entropy-insensitive for extremely dense matrices, $\mathcal{O}\big(\lg^2 n / \lg\lg n\big)$. In Section 7 we extend the result using $H + o(n^2)$ bits to d dimensions. Section 8 concludes and gives further research directions.

2. Basic Concepts

2.1. The One-Dimensional Case

The one-dimensional variant of the problem has long been studied. It can be modeled as a bitmap $B[1,n]$ with m 1s, corresponding to the positions of the points. The entropy of this bitmap is $H = \lg \binom{n}{m} = m \lg \frac{n}{m} + \mathcal{O}(m + \lg n)$. All the range counting, range reporting, and point selection queries can be solved in terms of two primitives: $\mathsf{rank}(B,i)$ is the number of 1s in $B[1,i]$, and $\mathsf{select}(B,j)$ is the position in B of the jth 1. For some applications it is also interesting to count and locate the 0s in the bitmap, so operations rank_0, rank_1, select_0 and select_1 are defined (assuming $\mathsf{rank} = \mathsf{rank}_1$ and $\mathsf{select} = \mathsf{select}_1$ by default). While $\mathsf{rank}_0(B,i) = i - \mathsf{rank}_1(B,i)$ is trivial, query $\mathsf{select}_0(B,j)$ needs to be solved independently of select_1.

Clark [11] and Munro [24] showed that all the rank and select queries can be solved in constant time using $n + o(n)$ bits of space, that is, B itself plus sublinear space. The structure using that $o(n)$ extra space is called an *index*, which operates by accessing a constant number of chunks of B. Golynski [15] showed that, in the model where the accesses to B are restricted to a black-box that returns any $\Theta(w) = \Theta(\lg n)$ consecutive bits in constant time, the index must use $\Omega\left(\frac{n \lg\lg n}{\lg n}\right)$ bits in order to achieve constant query time. He also designed an index that matched this lower bound.

Pagh [27] (see also Raman et al. [32]) provided a compressed representation of B that retrieves any $\Theta(\lg n)$ consecutive bits in constant time and requires $H + \mathcal{O}\left(\frac{n \lg\lg n}{\lg n}\right) = m \lg \frac{n}{m} + \mathcal{O}\left(m + \frac{n \lg\lg n}{\lg n}\right)$ bits. Combined with the index of Golynski [15], constant-time rank and select are supported within the same asymptotic space, $H + o(n)$ bits.

This $o(n)$-bit redundancy may dominate the entropy when m is much smaller than n. It was later shown that, by not separating the bitmap from the index, Golynski's lower bound can be broken, and $H + \mathcal{O}\left(\frac{n}{\mathrm{polylog}(n)}\right)$ bits can be achieved [17, 29, 31], but this is still too large if m is significantly smaller than n, e.g. $m = \mathcal{O}(n^\alpha)$ for a constant $0 < \alpha < 1$. Gupta et al. [22] removed this near-linear redundancy at the expense of non-constant query times. Their representation uses $m \lg \frac{n}{m} + \mathcal{O}\big(m \lg \frac{n}{m} / \lg m + m \lg\lg \frac{n}{m}\big) = m \lg \frac{n}{m} + o\left(m \lg \frac{n}{m}\right) + \mathcal{O}(m + \lg n)$ bits and answers rank and select queries in time $\mathcal{O}(\lg\lg m)$.

We prove now a technical lemma we will need later, related to an index having even more restricted access to the bitmap B. From now on, for

simplicity, we will omit floors and ceilings in our formulas.

Lemma 1. *Let $0 < \alpha \leq 1$ be a constant and $b = \Theta(\lg^\alpha n)$. Let bitmap $B[1,n]$ be stored in such a way that we can read chunks of the form $B[b\cdot(i-1)+1, b\cdot i]$ in constant time, for any i. Then we can perform* rank *and* select *in constant time using $\mathcal{O}\big(\frac{n \lg\lg n}{b}\big)$ bits of extra space, and this is optimal in general.*

Proof. We prove it only for $\alpha < 1$, as for $\alpha = 1$ the result is well known. We take any data structure achieving constant time and $\mathcal{O}\Big(\frac{n \lg\lg n}{\lg n}\Big)$ extra bits, say Golynski's [15], and adapt it to read aligned chunks of length b. The data structure uses several indexes and accesses B a constant number of times. Each such time, it reads a word of $w = \Theta(\lg n)$ consecutive bits of B, in order to either (a) count the number of 1s in a part of the word or (b) find the position of the kth 1 or 0 in a part of the word, using "universal tables" (i.e., small precomputed tables that do not depend on the content of B, only on b, and sum up to $o(n)$ bits of size).

We introduce an indirection when accessing such universal tables. Each word is covered by w/b chunks. For each chunk, we store the *summary* number of 1s in the chunk. This requires $\lg(b+1)$ bits, so the total space is $\mathcal{O}\big(\frac{n \lg b}{b}\big) = \mathcal{O}\big(\frac{n \lg\lg n}{b}\big)$. Moreover, in a RAM machine with word size w we can read all the summary numbers of the chunks covering any word in $\mathcal{O}(1)$ accesses, as they add up to $\frac{w \lg b}{b} = o(\lg n)$ bits. With these summary numbers we can index a universal table of $\mathcal{O}\big(2^{o(\lg n)}\mathrm{polylog}(n)\big) = o(n)$ bits, telling (a) the number of bits set up to any given chunk of the word, and (b) the chunk where the kth 0/1 of the word occurs. A final access to one b-bit chunk, with another universal table of $\mathcal{O}\big(2^b\mathrm{polylog}(n)\big) = o(n)$ bits, completes the query in constant time.

The lower bound comes directly from Golynski [15], who states that if one probes t bits and answers rank/select in constant time, then the index must be of size $\Omega\left(\frac{m \lg t}{t}\right)$. In the worst case $m = \Theta(n)$ and the query algorithms can access at most $t = \mathcal{O}(b)$ bits in constant time, so the index must use $\Omega\left(\frac{n \lg\lg n}{b}\right)$ bits in general (i.e., for any value of m). □

2.2. Two Dimensions

We will consider rectangular query ranges of the form $[i_1, i_2] \times [j_1, j_2] = \{(i,j),\ i_1 \leq i \leq i_2,\ j_1 \leq j \leq j_2\}$, where i_1 and i_2 are rows and j_1 and j_2 are columns in the grid. Over those ranges we define the queries

- $\mathsf{rank}(i_1, i_2, j_1, j_2)$ counts the number of points in the range; and
- $\mathsf{select}(i_1, i_2, j_1, j_2, k_1, k_2)$ gives the k_1th to the k_2th points in the range, in column-major or row-major order (this generalizes range reporting and point selection queries).

The general case is called a 4-sided query. A particular case, a 3-sided query, arises when one of the coordinates is always 1 or n. A 2-sided (also called dominance) query arises when two of the coordinates, one of row and one of column, is always 1 or n. A band query has 1 and n for either the row or the column coordinates. Finally, a 1-sided query has only one coordinate different from 1 or n.

Since $\mathsf{rank}(i_1, i_2, j_1, j_2) = \mathsf{rank}(1, i_2, 1, j_2) - \mathsf{rank}(1, i_1 - 1, 1, j_2) - \mathsf{rank}(1, i_2, 1, j_1 - 1) + \mathsf{rank}(1, i_1 - 1, 1, j_1 - 1)$, we study only 2-sided queries for rank, called $\mathsf{rank}(i, j) = \mathsf{rank}(1, i, 1, j)$. Also, for compliance with the existing literature, we prefer to study the queries in terms of selecting the kth point, $\mathsf{select}(i_1, i_2, j_1, j_2, k)$, and reporting any k points in a range, $\mathsf{report}(i_1, i_2, j_1, j_2, k)$. Our solutions, however, can actually be combined to solve the general $\mathsf{select}(i_1, i_2, j_1, j_2, k_1, k_2)$ query: Say we consider the points in column-major order. Then we (1) find the first and last points to report, $(i, j) = \mathsf{select}(i_1, i_2, j_1, j_2, k_1)$ and $(i', j') = \mathsf{select}(i_1, i_2, j_1, j_2, k_2)$; (2) if $j = j'$ the solution is simply $\mathsf{report}(i, i', j, j, k_2 - k_1 + 1)$; otherwise we produce the output with three calls: $\mathsf{report}(i, i_2, j, j, k)$ with $k = \mathsf{rank}(i, i_2, j, j)$, $\mathsf{report}(i_1, i', j', j', k')$ with $k' = \mathsf{rank}(i_1, i', j', j')$, and $\mathsf{report}(i_1, i_2, j + 1, j' - 1, k_2 - k_1 + 1 - k - k')$. (This returns the points in any order, otherwise we can simply use consecutive $\mathsf{select}(i_1, i_2, j_1, j_2, k)$ queries.)

Furthermore, we can focus on just band and 1-sided select queries. Assume that any $\mathsf{select}(i_1, i_2, j_1, j_2, k)$ query is valid, that is, it holds $k \leq \mathsf{rank}(i_1, i_2, j_1, j_2)$ (which can be checked beforehand). Then a band query, plus rank, are sufficient to solve a 4-sided query, $\mathsf{select}(i_1, i_2, j_1, j_2, k) = \mathsf{select}(i_1, i_2, 1, n, k + x)$ with $x = \mathsf{rank}(i_1, i_2, 1, j_1 - 1)$ if select delivers in column-major order, and analogously if in row-major order. Therefore 3-sided queries can also be converted into band queries.

Note that the resulting band query is "perpendicular", in the sense that the band is horizontal and the points are considered in column-major order, or vice versa. When both run in the same direction we say that the queries are "parallel", and they are simpler to handle. A parallel 3-sided query can be reduced to a 1-sided select query plus rank, for example $\mathsf{select}(1, i_2, j_1, j_2, k) = \mathsf{select}(1, i_2, 1, n, k + x)$ with $x = \mathsf{rank}(1, i_2, 1, j_1 - 1)$, if

the points are considered in column-major order. Therefore all the parallel queries can be reduced to (perpendicular) 1-sided queries (even parallel 1-sided queries can be converted to perpendicular ones in the same way). Note that a 4-sided query is always considered perpendicular.

Finally, our sublinear-sized indexes can be computed (or the algorithms trivially modified) for several rotations and reflections of the grid within the same asymptotic space. Therefore we can, without loss of generality, focus our study on the following queries:

- $\mathsf{rank}(i, j)$ is the number of points in $[1, i] \times [1, j]$;
- $\mathsf{select}(i_1, i_2, k)$ gives the kth point in the range $[i_1, i_2] \times [1, n]$, in column-major order (a perpendicular horizontal band query);
- $\mathsf{select}(i, k)$ gives the kth point in the range $[1, i] \times [1, n]$, in column-major order (a perpendicular horizontal 1-sided query);
- $\mathsf{report}(i_1, i_2, j_1, j_2, k)$ gives any k points in the range $[i_1, i_2] \times [j_1, j_2]$.

Just as in the one-dimensional case, we can identify a grid with a binary matrix, containing 1s at the positions of the m points and 0s elsewhere. Barbay et al. [3] propose a number of primitives on binary matrices. By using *wavelet trees* [20], they achieve the following result (we only mention the operations of interest for this paper, slightly adapting them to our purposes).

Lemma 2 ([3, Thm. 3]). *A binary matrix of σ rows ("labels") by n columns ("objects") with t 1s can be represented within $t \lg \sigma + o(t) \lg \sigma + \mathcal{O}(t + n)$ bits, so that queries* $\texttt{rel\_rnk}(i_1, i_2, j_1, j_2)$ *(number of points in $[i_1, i_2] \times [j_1, j_2]$), and* $\texttt{rel\_min\_obj\_maj}(i_1, i_2, j)$ *(first point, in object-major order, in $[i_1, i_2] \times [j, n]$), are answered in time $\mathcal{O}(\lg \sigma / \lg \lg n)$. Furthermore, query* $\texttt{rel\_acc}(i_1, i_2, j_1, j_2)$ *(giving all the k points in $[i_1, i_2] \times [j_1, j_2]$), is answered in time $\mathcal{O}((k + 1) \lg \sigma / \lg \lg n)$. Finally, query* $\texttt{rel\_sel\_lab\_maj}(i, k, j_1, j_2)$ *(kth point, in label-major order, in $[i, \sigma] \times [j_1, j_2]$) requires time $\mathcal{O}(\lg \sigma)$, and query* $\texttt{rel\_sel\_obj\_maj}(i_1, i_2, k, j)$ *(kth point, in object-major order, in $[i_1, i_2] \times [j, n]$) requires time $\mathcal{O}(\lg \sigma \lg n / \lg \lg n)$.*

Table 1 gives in detail the complexities achieved for our operations. The first lines report the results of two simple solutions we develop next by easily building on previous work. Then we give our main solutions. We remark

Source	Space	rank time	report time
Lem. 3+[6]	$H + o(H) + \mathcal{O}(m + \lg n)$	$\lg m / \lg\lg m$	$\lg m / \lg\lg m$
Lem. 4+[3]	$H + o(H) + \mathcal{O}(m + \lg n)$	$\lg m / \lg\lg m$	$\lg m / \lg\lg m$
Thm. 1	$H + \mathcal{O}\left(\frac{n^2 \lg\lg n}{\lg^{1/4} n}\right)$	1	1
Thm. 2	$H + o(H) + \mathcal{O}(m \lg\lg\lg m)$	$\lg \frac{n^2}{m} / \lg\lg n$	$\lg \frac{n^2}{m} / \lg\lg n$
Thm. 3	$H + o(H)$	$\lg n / \lg\lg n$	$\lg^2 n / \lg\lg n$

Source	select time	
	General	1-sided or parallel
Lem. 3+[6]	$\lg^2 m / \lg\lg m$	$\lg^2 m / \lg\lg m$
Lem. 4+[3]	$\lg m$ or $\lg^2 m / \lg\lg m$	$\lg m$ or $\lg^2 m / \lg\lg m$
Thm. 1	$\lg n$	$\lg\lg n$
Thm. 2	$\lg n$ or $\lg n + \lg^2 \frac{n^2}{m} / \lg\lg n$	$\lg \frac{n^2}{m}$ or $\lg^2 \frac{n^2}{m} / \lg\lg n$
Thm. 3	$\lg^2 n / \lg\lg n$	$\lg^2 n / \lg\lg n$

Table 1: Space and time complexities of entropy-compressed grid representations. The "or" case depends on using row-major or column-major order to traverse the points. The times for report are to be multiplied by $k + 1$ in order to retrieve k points. General select times refer to our perpendicular band queries, which can simulate any other, whereas the next column refers to the queries that are simulated with perpendicular 1-sided queries. The time complexities of Theorem 2 are simplified for the case $m = \mathcal{O}\left(\frac{n^2}{\lg^{1/5} n}\right)$, where it achieves $H + o(H)$ bits of space.

that the space of Theorem 2 is $H + o(H)$ if $m = \mathcal{O}\left(\frac{n^2}{\lg^{1/5} n}\right)$, where we obtain entropy-sensitive complexity, and the space of Theorem 1 is $H + o(H)$ whenever $\min(m, n^2 - m) = \omega\left(\frac{n^2}{\lg^{1/5} n}\right)$, where we achieve $\mathcal{O}(1)$ time for counting and reporting.

When regarding the grids as binary matrices, it makes sense to consider the complementary operations, so that we are also interested in counting, reporting and selecting 0s in a range. While this is trivial for rank queries because $\mathsf{rank}_0(i_1, i_2, j_1, j_2) = (i_2 - i_1 + 1)(j_2 - j_1 + 1) - \mathsf{rank}_1(i_1, i_2, j_1, j_2)$, the other operations need separate procedures. We obtain the same (Theorems 1 and 3) or slightly worse (Theorem 2) results for those. Finally, our results adapt smoothly to rectangular grids of size $n_r \times n_c$ by replacing every n by $\sqrt{n_r n_c}$ in the table.

3. Two Simple Solutions

We develop now two simple solutions by building on previous work. Then we improve those results in the rest of the paper.

Assume we map the $n \times n$ grid with m points into an $m \times m$ grid with m points. This is done by removing empty rows and columns, and duplicating rows or columns having more than one point, so that in the mapped matrix there is exactly one point per row and per column. This duplication can be done so as to be consistent with row- and column-major orderings for select.

Two bitmaps, $R[1, n+m+1]$ and $C[1, n+m+1]$, represent the mapping from the original matrix, for rows and columns respectively. To build R, we traverse the original matrix in row-major order, appending a 0 each time we start a row and a 1 each time we find a point, and add a final 0. Then the original row i starts at row $\mathsf{select}_0(R, i) - i + 1$ and finishes at row $\mathsf{select}_0(R, i+1) - i - 1$ of the mapped matrix. The column mapping C is analogous. With these operations we can easily map any query range to the mapped matrix.

We use Gupta et al.'s representation [22] for R and C, which solves the queries in time $\mathcal{O}(\lg \lg m)$ and requires $2m \lg \frac{n+m}{m} + o\left(m \lg \frac{n+m}{m}\right) + \mathcal{O}(m + \lg n) = 2m \lg \frac{n}{m} + o\left(m \lg \frac{n}{m}\right) + \mathcal{O}(m + \lg n)$ bits for the two bitmaps.

Now for the mapped grid we can use Bose et al.'s representation [6], which solves rank in time $\mathcal{O}(\lg m / \lg \lg m)$, report in time $\mathcal{O}((k+1) \lg m / \lg \lg m)$, and select in time $\mathcal{O}\left(\lg^2 m / \lg \lg m\right)$ via binary searches on rank.

As for space, the structure requires $m \lg m + o(m \lg m)$ bits, which added to the space for R and C gives $m \lg \frac{n^2}{m} + o\left(m \lg \frac{n^2}{m}\right) + \mathcal{O}(m + \lg n)$ bits. This

is $H + o(H) + \mathcal{O}(m + \lg n)$.

Lemma 3. *An $n \times n$ grid with m points can be represented within $H + o(H) + \mathcal{O}(m + \lg n)$ bits of space, where $H = \lg \binom{n^2}{m}$, so that query* $\mathsf{rank}(i, j)$ *is computed in $\mathcal{O}(\lg m / \lg\lg m)$ time,* $\mathsf{report}(i_1, i_2, j_1, j_2, k)$ *performs in time $\mathcal{O}((k+1) \lg m / \lg\lg m)$, and* $\mathsf{select}(i_1, i_2, k)$ *and* $\mathsf{select}(i, k)$ *are supported in $\mathcal{O}\big(\lg^2 m / \lg\lg m\big)$ time.*

An alternative is to use Barbay et al.'s representation [3] for the mapped grid. The space of this structure is $m \lg m + o(m \lg m) + \mathcal{O}(m + n)$, but the last term is not necessary in our simplified matrix with exactly one point per column and per row. Then rank is solved using `rel_rnk` in time $\mathcal{O}(\lg m / \lg\lg m)$, report is solved using `rel_acc` in time $\mathcal{O}((k+1) \lg m / \lg\lg m)$, and both select queries are solved using either `rel_sel_lab_maj` or `rel_sel_obj_maj`, in time $\mathcal{O}(\lg m)$ or $\mathcal{O}\big(\lg^2 m / \lg\lg m\big)$, depending on the direction of the query.

Lemma 4. *An $n \times n$ grid with m points can be represented within $H + o(H) + \mathcal{O}(m + \lg n)$ bits of space, where $H = \lg \binom{n^2}{m}$, so that query* $\mathsf{rank}(i, j)$ *is computed in $\mathcal{O}(\lg m / \lg\lg m)$ time, and* $\mathsf{report}(i_1, i_2, j_1, j_2, k)$ *performs in time $\mathcal{O}((k+1) \lg m / \lg\lg m)$. In one direction (that can be chosen),* $\mathsf{select}(i_1, i_2, k)$ *and* $\mathsf{select}(i, k)$ *queries are supported in $\mathcal{O}(\lg m)$ time; in the other they take $\mathcal{O}\big(\lg^2 m / \lg\lg m\big)$ time.*

Our next developments aim at two goals. First, we show that it is possible to obtain a *fully compressed* representation, that is, using strictly $H + o(H)$ bits. Second, we show that some operations can be speeded up, taking less time when the entropy of the matrix is higher.

4. A Fast Compressed Representation with Sublinear Redundancy

We first describe a solution using $n^2 + o(n^2)$ bits, and then convert it into one using $H + o(n^2)$ bits. Using this $o(n^2)$-size index on top of the compressed grid, we handle various operations in constant time.

4.1. Constant-Time Rank

The matrix is first subdivided into *superblocks* of size $s \times s$, $s = \lg^2 n$. Each superblock is in turn subdivided into *blocks* of size $b \times b$, $b = \sqrt{\frac{\lg n}{2}}$.

The n^2 bits of the matrix will be stored block-wise, that is, the $b^2 = \frac{\lg n}{2}$ bits of each block will be stored contiguously.

For each superblock in the matrix, we store the rank values at all the positions of the rightmost column and bottom row of the superblock. In other words, we store all $\mathsf{rank}(i, s \cdot j_s)$ and $\mathsf{rank}(s \cdot i_s, j)$ values, for $1 \leq i, j \leq n$ and $1 \leq i_s, j_s \leq n/s$. This requires $\mathcal{O}\left(\frac{n^2 \lg n}{\lg^2 n}\right) = o(n^2)$ bits. For each block within each superblock, we store the *local* (i.e., within its superblock) rank values at all the positions of the rightmost column and bottom row of the block. If we call rank_s those local rank values, what we store are all $\mathsf{rank}_s(i', b \cdot j_b)$ and $\mathsf{rank}_s(b \cdot i_b, j')$ values, for $1 \leq i', j' \leq s$ and $1 \leq i_b, j_b \leq s/b$. This requires $\mathcal{O}\left(\frac{n^2 \lg \lg n}{\sqrt{\lg n}}\right) = o(n^2)$ bits.

This gives enough information to compute $\mathsf{rank}(i, j)$ in constant time. Let $i = s \cdot i_s + i_{rs}$ and $j = s \cdot j_s + j_{rs}$, so that $s \cdot i_s$ and $s \cdot j_s$ are the projections of i and j to the last superblock-aligned row and column, and $0 \leq i_{rs}, j_{rs} < s$ are the local positions within their superblock. Similarly, let $i_{rs} = b \cdot i_b + i_{rb}$ and $j_{rs} = b \cdot j_b + j_{rb}$, with $0 \leq i_{rb}, j_{rb} < b$ the projections into, and local coordinates within, the blocks. Then it is easy to verify that

$$\begin{aligned}
\mathsf{rank}(i, j) &= \mathsf{rank}_b(i_{rb}, j_{rb}) \\
&+ \mathsf{rank}_s(i, b \cdot j_b) + \mathsf{rank}_s(b \cdot i_b, j) - \mathsf{rank}_s(b \cdot i_b, b \cdot j_b) \\
&+ \mathsf{rank}(i, s \cdot j_s) + \mathsf{rank}(s \cdot i_s, j) - \mathsf{rank}(s \cdot i_s, s \cdot j_s),
\end{aligned}$$

where $\mathsf{rank}_b(i_{rb}, j_{rb})$ is the local rank value within its block. All the rank and rank_s values in the formula are stored, whereas $\mathsf{rank}_b(i_{rb}, j_{rb})$ will be solved with a universal table: As there are only $2^{b^2} = \sqrt{n}$ different blocks, we can store all the answers to all possible rank_b queries within $\mathcal{O}(\sqrt{n}\,\mathrm{polylog}(n)) = o(n)$ bits. Since we can read at once the $b^2 = \mathcal{O}(\lg n)$ bits of the block (stored contiguously as explained), we can look up a table entry in constant time.

4.2. Constant-Time Report

We first solve a subproblem that might have independent interest. Given a row range $[i_1, i_2]$ and a column j, $nextCol(i_1, i_2, j)$ is the smallest column number $j' > j$ that is nonempty (i.e., contains a 1) in the range $[i_1, i_2]$. We now show how to support this query in constant time and $o(n^2)$ extra bits.[2]

[2]This is simplified from the conference version [13] thanks to an anonymous reviewer.

We divide the rows into *chunks* of $r = \lg^{1/4} n$ rows. For each column j, $1 \leq j \leq n$, we compute array $A_j[1, n/r]$, so that $A_j[i_r] = nextCol(r \cdot (i_r - 1) + 1, r \cdot i_r, j)$ is the next nonempty column in the chunk i_r. We do not store the arrays A_j themselves, but just a range minimum query (RMQ) data structure on each. Such queries find the position of the minimum in any range of the array, and can be implemented to answer in constant time and taking $2n/r + o(n/r)$ bits each, without the need to access A_j [14]. Over the n columns, the space adds up to $\mathcal{O}(n^2/r) = o(n^2)$ bits.

Now consider a chunk-aligned query $nextCol(r \cdot (i_1 - 1) + 1, r \cdot i_2, j)$. We find the position i_r of the minimum in $A_j[i_1, i_2]$, and know that the answer is to be found within chunk i_r. If, instead, the query is not chunk-aligned, then it can be decomposed into a (possibly empty) chunk-aligned band, plus a within-chunk band above it and a within-chunk band below it. Otherwise, the query is completely contained in a chunk. In all cases, since the RMQ structures narrow down the search on the chunk-aligned band to a single chunk, the query is reduced to at most three within-chunk queries, to return the minimum of their answers.

Now, confined within a chunk of r rows, we consider bit vectors $B(i_1, i_2)$, $1 \leq i_1 \leq i_2 \leq r$, such that $B(i_1, i_2)$ is the *or* of rows from i_1 to i_2, $B(i_1, i_2)[j] = M[i_1, j]$ *or* $M[i_1 + 1, j]$ *or* $\ldots$ *or* $M[i_2, j]$ where M denotes the binary matrix. We cannot explicitly store all these vectors, as the space would be $\omega(n^2)$. However, we store the rank and select *indexes* for each such bit vector. To simulate access to the virtual bit vector $B(i_1, i_2)$, we use our $b \times b$ blocks of M stored contiguously, in order to provide in constant time any $\Theta(\sqrt{\lg n})$ consecutive bits of any $B(i_1, i_2)$. This is done with a universal table of size $\mathcal{O}(\sqrt{n}\,\mathrm{polylog}(n)) = o(n)$ bits that, for each possible $b \times b$ block and values $1 \leq i_1 \leq i_2 \leq b$, stores a bitmap *or*-ing rows i_1 to i_2 of the block.

By Lemma 1, since we can simulate access to contiguous regions of $B(i_1, i_2)$ of length $\Theta(\sqrt{\lg n})$, we can achieve constant time for rank and select using extra indexes of $\mathcal{O}\left(\frac{n \lg\lg n}{\sqrt{\lg n}}\right)$ bits. With these operations any within-chunk query is solved in constant time, as it is equivalent to finding the first 1 in $B(i_1, i_2)[j + 1, n]$, $\mathsf{select}(B(i_1, i_2), \mathsf{rank}(B(i_1, i_2), j) + 1)$.

As there are $\mathcal{O}\left(\frac{n}{\lg^{1/4} n}\right)$ chunks, each storing $\mathcal{O}\left((\lg^{1/4} n)^2\right)$ indexes for $B(i_1, i_2)$, the total space is $\mathcal{O}\left(\frac{n}{\lg^{1/4} n} \cdot \sqrt{\lg n} \cdot \frac{n \lg\lg n}{\sqrt{\lg n}}\right) = o(n^2)$ bits.

Once *nextCol* is solved, it is easy to address $\mathsf{report}(i_1, i_2, j_1, j_2, k)$ queries. We store one-dimensional rank and select indexes for every column of the

matrix. As already explained, their extra space adds up to $\mathcal{O}\left(\frac{n \lg\lg n}{\sqrt{\lg n}}\right) = o(n)$ bits per column as we can access only $\Theta(\sqrt{\lg n})$ contiguous bits of any column. The first points to report are at column $j = nextCol(i_1, i_2, j_1 - 1)$. With one-dimensional rank and select on column j, we can report the points at rows $[i_1, i_2]$ of that column, each in constant time. We go on with $j = nextCol(i_1, i_2, j)$, and so on, until either $j > j_2$ or we have reported k points. Thus the query takes time $\mathcal{O}(k + 1)$.

4.3. Select Queries

For $\mathsf{select}(i_1, i_2, k)$ we binary search, using rank, the position of the kth point in $\mathcal{O}(\lg n)$ time. We can do better for the simpler $\mathsf{select}(i, k)$ query. We have already stored the rank values at the rightmost columns of the superblocks. Assume these values are organized row-wise, and stored in one y-fast trie data structure [33] per row. This sums to $\mathcal{O}\left(\frac{n \lg n}{\lg^2 n}\right) = o(n)$ bits per row. The trie for row i permits finding the superblock column containing the kth point in $[1, i] \times [1, n]$, in $\mathcal{O}(\lg\lg n)$ time (by finding the predecessor of k). Now a binary search over the $s = \lg^2 n$ values $\mathsf{rank}(1, i, 1, j)$ for the columns j inside the superblock column found gives, in another $\mathcal{O}(\lg\lg n)$ time, the precise column. Finally, one-dimensional rank and select on the column give the position of the kth point. Thus the time is $\mathcal{O}(\lg\lg n)$.

4.4. Entropy-Bounded Space

We have assumed the $b \times b$ blocks are explicitly stored. Instead, we can replace them by a (c, o) pair, just as Pagh [27] does for one-dimensional bit vectors. Let a block contain c 1s. Then its *class* is c and its *offset* o is an (arbitrary) identifier of this particular $b \times b$ block among all the different blocks of class c. A universal table indexed by c and o storing the contents of all the possible bit vectors of each class has $\sum_{0 \le c \le b} \binom{b}{c} = 2^b = \sqrt{n}$ entries and takes $\mathcal{O}(\sqrt{n} \lg n) = o(n)$ bits, and recovers any block content in constant time from its (c, o) code.

Each c value is stored in $\lg(b^2 + 1) = \mathcal{O}(\lg\lg n)$ bits, adding up to $\mathcal{O}\left(\frac{n^2 \lg\lg n}{\lg n}\right) = o(n^2)$ bits in total. The number of bits required for all the o fields, assuming the ith block contains m_i bits set, is $\sum_i \lceil \lg \binom{b^2}{m_i} \rceil \le \lg \binom{n^2}{m} + \mathcal{O}\left(\frac{n^2}{\lg n}\right)$ [27, Lem. 4.1]. Finally, we also need pointers to find an o field in constant time, as these have variable-length representations. These pointers can also be represented within $\mathcal{O}\left(\frac{n^2 \lg\lg n}{\lg n}\right)$ bits [27].

Theorem 1. *An $n \times n$ binary matrix with m 1s and entropy $H = \lg \binom{n^2}{m}$ can be represented within $H + \mathcal{O}\Big(\frac{n^2 \lg\lg n}{\lg^{1/4} n}\Big)$ bits, so that query* $\mathsf{rank}(i,j)$ *is computed in $\mathcal{O}(1)$ time,* $\mathsf{report}(i_1, i_2, j_1, j_2, k)$ *performs in time $\mathcal{O}(k+1)$,* $\mathsf{select}(i_1, i_2, k)$ *is supported in $\mathcal{O}(\lg n)$ time, and* $\mathsf{select}(i,k)$ *takes $\mathcal{O}(\lg\lg n)$ time.*

4.5. Extensions

We can define the complementary queries, where 0s are considered instead of 1s. As explained, this is immediate for rank but not for report nor select. However, it is not hard to see that we can support these complementary queries as well, by adding other similar $o(n^2)$ bits of space corresponding to the complemented matrix, that is, asymptotically for free. As explained, we can also support the select variants where rows and columns are exchanged, within $o(n^2)$ additional space.

In order to handle rectangular grids of size $n_r \times n_c$, we can define square superblocks of side $s = \lg^2(n_r n_c)$, square blocks of side $b = \sqrt{\lg(n_r n_c)/2}$ and chunks of $r = \lg^{1/4}(n_r n_c)$ rows. We obtain $H + \mathcal{O}\Big(\frac{n_r n_c \lg\lg(n_r n_c)}{\lg^{1/4}(n_r n_c)}\Big) = H + o(n_r n_c)$ bits of space, and the same operation times of Theorem 1, replacing every n by $\sqrt{n_r n_c}$.

If the two sides of the rectangle are very different, say $n_r = o(s) = o(\lg^2 n_c)$ (thus $\lg n_r = \mathcal{O}(\lg\lg n_c)$ and $\lg(n_r n_c) = \Theta(\lg n_c)$), we cannot use those superblock sizes anymore. Instead, we can use just one wavelet tree of Lemma 2 with n_r labels, which will answer rank queries in time $\mathcal{O}(\lg n_r / \lg\lg(n_r n_c)) = \mathcal{O}(1)$, report queries in time $\mathcal{O}((k+1)\lg n_r / \lg\lg(n_r n_c)) = \mathcal{O}(k+1)$, and queries select in time $\mathcal{O}(\lg n_r)$ or $\mathcal{O}\big(\lg^2 n_r / \lg\lg(n_r n_c)\big)$, both $\mathcal{O}(\lg\lg n_c)$. Thus the time complexities are retained. The space of this wavelet tree is $m \lg n_r + o(m \lg n_r) + \mathcal{O}(m + n_c)$. The last term is due to a bitmap of length $m + n_c$, with m bits set, which is stored in plain form [3]. Instead, we use Raman et al.'s compressed representation [32], which retains constant rank and select time and uses $m \lg \frac{m+n_c}{m} + \mathcal{O}\Big(m + \frac{n_c \lg\lg n_c}{\lg n_c}\Big) = m \lg \frac{n_c}{m} + \mathcal{O}\Big(m + \frac{n_c \lg\lg n_c}{\lg n_c}\Big)$ bits. Added to the space of the wavelet tree, we have $m \lg \frac{n_r n_c}{m} + o\left(m \lg \frac{n_r n_c}{m}\right) + \mathcal{O}\Big(m + \frac{n_c \lg\lg n_c}{\lg n_c}\Big)$ bits. This is $H + o(n_r n_c)$.[3]

[3]Moreover, it can be shown to be $H + \mathcal{O}\Big(\frac{n_r n_c \lg\lg(n_r n_c)}{\lg^{1/4}(n_r n_c)}\Big)$ for the range of values of m

5. Towards a Fully-Compressed Representation

Our compressed representation achieves entropy-bounded space for the matrix itself, but the extra space is $o(n^2)$. This may dominate the entropy bound H. In this section we get much closer to using $H + o(H)$ bits. The key to achieving indexes sublinear in H is to adapt the partitioning into superblocks and blocks to the number of bits set in the matrix. As now the blocks will be much larger, we cannot handle them with universal tables, but will make use of the wavelet trees of Lemma 2. The price will be a superconstant time for all queries.

5.1. Rank Query

We first divide the matrix into superblocks of size $s \times s$, where now $s = \frac{n^2 \lg m}{m}$ (assume for now $m = \Omega(n \lg m)$; we consider the other case in Section 5.4). The superblocks are further divided into blocks of size $b \times b$, for $b = \frac{n^2 \lg \lg m}{m}$. Just as for Section 4.1, we store absolute ranks at the borders of superblocks and local ranks at the borders of blocks. As the former require $\lg m$ bits to be represented, they add up to $\mathcal{O}\left(\frac{n^2}{s} \lg m\right) = \mathcal{O}(m)$ bits. The latter require $\lg s^2$ bits per value, adding up to $\mathcal{O}\left(\frac{n^2}{b} \cdot \lg s^2\right) = \mathcal{O}\left(\frac{m}{\lg \lg m} \cdot \lg \frac{n^2 \lg m}{m}\right) = \mathcal{O}\left(\frac{m}{\lg \lg m}\left(\lg \frac{n^2}{m} + \lg \lg m\right)\right) = o\left(m \lg \frac{n^2}{m}\right) + \mathcal{O}(m)$.

As before, the problem is reduced to supporting local rank within a block of size b^2. We store each whole column of blocks $[b \cdot (j_b - 1) + 1, b \cdot j_b] \times [1, n]$, for $1 \le j_b \le n/b$, using the wavelet tree of Lemma 2. This is regarded as having n objects (the ith object represents row i) and b labels (the jth label represents column $b \cdot (j_b - 1) + j$). Say that column of blocks j_b contains m_{j_b} bits set, then its wavelet tree requires $m_{j_b} \lg b + o(m_{j_b} \lg b) + \mathcal{O}(m_{j_b} + n)$ bits. Added over all the columns of blocks, this is $m \lg b + o(m \lg b) + \mathcal{O}(m + n^2/b) = m \lg \frac{n^2}{m} + o\left(m \lg \frac{n^2}{m}\right) + \mathcal{O}(m \lg \lg \lg m)$.

The wavelet tree answers `rel_rnk` queries in time $\mathcal{O}(\lg b / \lg \lg n) = \mathcal{O}\left(\lg \frac{n^2}{m} / \lg \lg n\right)$. The local $\mathsf{rank}_b(i, j)$ value within a block is thus computed using two `rel_rnk` queries on the wavelet tree representing the column that contains the block: Let i' be the top row of the block of interest, then

---

where Theorem 1 will be used, $m = \omega\left(\frac{n_r n_c}{\lg^{1/5}(n_r n_c)}\right)$: In this range it holds $m \lg \frac{n_r n_c}{m} = \Omega(m \lg \lg n_c)$ and the $o(m \lg n_r)$ bits of the wavelet tree are $o(m \lg \lg n_c)$.

$\mathsf{rank}_b(i,j) = \texttt{rel\_rnk}(j, i'-1+i) - \texttt{rel\_rnk}(j, i'-1)$. The result is analogous if we choose to represent rows of blocks with the wavelet trees.

5.2. Select Queries

Let us first consider (horizontal) band queries, assuming that wavelet trees represent columns of blocks. As we have stored all the values $\mathsf{rank}(i, s \cdot j_s)$, $\mathsf{rank}(s \cdot i_s, j)$, and $\mathsf{rank}_s(i, b \cdot j_b)$, we can compute any $\mathsf{rank}(i_1, i_2, 1, b \cdot j_b)$ in constant time. Thus we can binary search for the column of blocks where the kth point of $[i_1, i_2] \times [1, n]$ lies. This takes time $\mathcal{O}\big(\lg \frac{n}{b}\big)$.

Let j_b be the column of blocks found, then the local rank of the (globally) kth point, within block-column j_b, is $k' = k - \mathsf{rank}(i_1, i_2, 1, b \cdot (j_b - 1))$. Now we can use the wavelet tree of the j_bth block of columns to find the k'th point, in label-major order, between objects i_1 and i_2, with operation `rel_sel_lab_maj`, in $\mathcal{O}(\lg b)$ time. Overall, the query takes time $\mathcal{O}(\lg n)$.

The situation is more complicated if wavelet trees represent rows of blocks (or, symmetrically, the query band is vertical, but we are sticking to horizontal bands). After finding the column of blocks j_b where the answer lies, we refine the search to find its exact column. We binary search within columns $[b \cdot (j_b - 1) + 1, b \cdot j_b]$ using $\mathsf{rank}(i_1, i_2, 1, j)$, for $b \cdot (j_b - 1) < j \leq b \cdot j_b$. This rank is not constant-time because we are not in borders of blocks. Hence the time rises to $\mathcal{O}\big(\lg^2 b / \lg\lg n\big) = \mathcal{O}\Big(\lg^2 \frac{n^2}{m} / \lg\lg n\Big)$.

Once we know the column j, we must find the k''th point in it, within rows $[i_1, i_2]$, for $k'' = k - \mathsf{rank}(i_1, i_2, 1, j-1)$. We first search the column for the block row i_b where k'' lies. This can be done in time $\mathcal{O}\big(\lg \frac{n}{b}\big)$ because the values $\mathsf{rank}(b \cdot i_b, j)$ are precomputed.

Finally, when we are confined within a single column of a block, we report the correct point in $\mathcal{O}(\lg b)$ time using `rel_sel_lab_maj` on the wavelet tree of the column of blocks (note that the area is of width 1, so the query is correct even if the wavelet tree orders points in column-major order). Overall, the query time is $\mathcal{O}\Big(\lg n + \lg^2 \frac{n^2}{m} / \lg\lg n\Big)$.

Now we consider the (horizontal) 1-sided select queries. We use the same procedure of band queries, but we do better when searching for j_b. For each row, we arrange the n/s superblock ranks in a y-fast trie, so as to pay $\mathcal{O}(\lg\lg m)$ time to find the superblock (note that the trie stores values up to m), and then binary search for j_b inside its superblock in time $\mathcal{O}\big(\lg \frac{s}{b}\big) = \mathcal{O}(\lg\lg m)$. This trie requires $\mathcal{O}\Big(\frac{n^2}{s} \lg m\Big) = \mathcal{O}(m)$ bits of space. Therefore,

in the easy case where the direction of the query is perpendicular to that of wavelet trees, 1-sided queries require time $\mathcal{O}\Big(\lg\lg m + \lg\frac{n^2}{m}\Big)$.

For the harder case, we also use a second set of y-fast tries to speed up the binary searches in the values $\mathsf{rank}(b \cdot i_b, j)$. For each column j, we store the values $\mathsf{rank}(\lg m \cdot b \cdot i_b, j)$ in a y-fast trie. We first search the y-fast trie in time $\mathcal{O}(\lg\lg m)$, and then binary search the area of $\lg m$ blocks found with the y-fast trie, again in time $\mathcal{O}(\lg\lg m)$. The n tries, one per column, require $\mathcal{O}\Big(\frac{n^2 \lg m}{b \lg m}\Big) = o(m)$ bits. Thus the time is $\mathcal{O}\Big(\lg\lg m + \lg\frac{n^2}{m} + \lg^2\frac{n^2}{m}/\lg\lg n\Big)$.

5.3. Range Reporting

Let us assume that we have stored rows of blocks in wavelet trees (the other case is analogous because reporting queries are 4-sided and we report in no particular order). We start considering $\mathsf{report}(i_1, i_2, j_1, j_2, k)$ queries that span an integral number of rows of blocks. We first need to find the next column after $j_1 - 1$ that is nonempty in the range $[i_1, i_2]$. We use the same RMQ-based idea of Section 4.2, using blocks of height b instead of chunks of height r. We store the RMQ structures of the arrays A_j corresponding to blocks of rows, for a total space of $\mathcal{O}(n^2/b) = o(m)$ bits. Instead of the virtual bitmaps $B(i_1, i_2)$ used for ranges $[i_1, i_2]$ within a block of rows, to find the next 1 in the band $[i_1, i_2] \times [j_1, n]$ we use the horizontally arranged wavelet trees. We find this 1 using `rel_min_obj_maj`, in $\mathcal{O}(\lg b/\lg\lg n)$ time.

Say such a query gives column j as the next one containing points. Now we must find all the 1s in column j before proceeding to the next one. Those 1s within the first block from global row i_1 are easily found with `rel_acc` in the wavelet tree of the block, each in time $\mathcal{O}(\lg b/\lg\lg n)$. In order to find the next block downwards containing points in column j, we store a signature bit vector $B_j[1, n/b]$ for each column j, so that $B_j[i_b] = 1$ iff there is a 1 in the range $[b \cdot (i_b - 1) + 1, b \cdot i_b] \times [j, j]$ of the matrix. Using rank and select on the B_j vector, we find the next block downwards that has a 1 in the current column, in constant time. All the points in column j mod b of that block are then reported using `rel_acc` on the object j of the wavelet tree that represents that row. Bit vectors B_j require $\mathcal{O}(n^2/b) = o(m)$ bits in total.

Thus the total time to report k points is $\mathcal{O}((k+1)\lg b/\lg\lg n) = \mathcal{O}\Big((k+1)\lg\frac{n^2}{m}/\lg\lg n\Big)$. If the query is not aligned to rows of blocks, it may have one unaligned band above and one below the block-aligned part. Then, in addition to the points reported by the procedure described, we use

`rel_acc` on the wavelet trees of the (one or two) partially covered rows of blocks. Similarly, if the query is totally contained in a block of rows, it is directly solved with a single wavelet tree. The time complexity is maintained.

5.4. The Final Result

A missing piece is to cover the case $m = o(n \lg m) = o(n \lg n)$, where our partition into superblocks of size s and blocks of size b does not work anymore because it requires $s = \omega(n)$. When the matrix is so sparse we have $\lg \frac{n^2}{m} = \Theta(\lg n)$, and thus we can just use the simple Lemma 4, whose times are within the general times we have obtained for denser matrices.

Summing up, the space is $m \lg \frac{n^2}{m} + o\left(m \lg \frac{n^2}{m}\right) + \mathcal{O}(m \lg\lg\lg m + \lg n)$. By Eq. (1), it holds $H = m \lg \frac{n^2}{m} + (n^2 - m) \lg \frac{n^2}{n^2-m} + \mathcal{O}(\lg n) \geq m \lg \frac{n^2}{m}$, therefore the space can be written as $H + o(H) + \mathcal{O}(m \lg\lg\lg m + \lg n)$. The last term, $\lg n$, is relevant only if $m = \mathcal{O}(1)$, in which case we can just store the points in compressed form and solve all the queries in constant time by traversing them all. Therefore, we can safely remove this term.

Theorem 2. *An $n \times n$ binary matrix with m 1s can be stored in $H + o(H) + \mathcal{O}(m \lg\lg\lg m)$ bits, so that query $\mathsf{rank}(i, j)$ is computed in $\mathcal{O}\left(\lg \frac{n^2}{m} / \lg\lg n\right)$ time and $\mathsf{report}(i_1, i_2, j_1, j_2, k)$ in time $\mathcal{O}\left((k+1) \lg \frac{n^2}{m} / \lg\lg n\right)$. In one direction (that can be chosen), $\mathsf{select}(i_1, i_2, k)$ is computed in $\mathcal{O}(\lg n)$ time, and $\mathsf{select}(i, k)$ in time $\mathcal{O}\left(\lg\lg m + \lg \frac{n^2}{m}\right)$. In the other direction, $\mathsf{select}(i_1, i_2, k)$ requires $\mathcal{O}\left(\lg n + \lg^2 \frac{n^2}{m} / \lg\lg n\right)$ time, and $\mathsf{select}(i, k)$ requires $\mathcal{O}\left(\lg\lg m + \lg \frac{n^2}{m} + \lg^2 \frac{n^2}{m} / \lg\lg n\right)$ time.*

In the final construction, we will make use of Theorem 2 for $m = \mathcal{O}\left(\frac{n^2}{\lg^{1/5} n}\right)$. In this case, its time complexities are considerably simplified because $\lg \frac{n^2}{m}$ becomes $\Omega(\lg\lg n)$, and thus $\lg\lg m$ is absorbed by $\mathcal{O}\left(\lg \frac{n^2}{m}\right)$ and this term in turn is absorbed by $\mathcal{O}\left(\lg^2 \frac{n^2}{m} / \lg\lg n\right)$. The space is also simplified, because it holds $m \lg\lg\lg m = o\left(m \lg \frac{n^2}{m}\right) = o(H)$, and thus the total space becomes fully compressed, $H + o(H)$.

5.5. Extensions

If we want to operate on 0s instead of on 1s, some queries become costlier. We need no further space nor time for rank, as explained. For all band select queries we can essentially use the procedure described for band queries when the wavelet trees are parallel to the band, as it is entirely based on binary searches on rank values. The only difference is that we cannot use `rel_sel_lab_maj`, but must resort to binary search inside the block using `rel_rnk`. This does not change the complexity $\mathcal{O}\Big(\lg n + \lg^2 \frac{n^2}{m} / \lg\lg n\Big)$. The 1-sided queries can be speeded up with y-fast tries counting 0s. Since these may store values up to n^2, they will answer in time $\mathcal{O}(\lg\lg n)$ and will be sampled every $v = \frac{n^2 \lg n}{m}$ rows or columns, so that they take $\mathcal{O}\Big(\frac{n^2 \lg n}{v}\Big) = \mathcal{O}(m)$ bits. The y-fast trie on superblock columns will store one value every $v/s = \frac{\lg n}{\lg m}$ superblocks, and thus the total search time will be $\mathcal{O}(\lg\lg n)$. The second y-fast trie on block rows that are multiples of $\lg m$ will store one value out of $\frac{v}{b \lg m} = \frac{\lg n}{\lg m \lg\lg m}$, so the search time is the same. Overall, 1-sided select queries of 0s take time $\mathcal{O}\Big(\lg\lg n + \lg^2 \frac{n^2}{m} / \lg\lg n\Big)$. Under the assumption $m = \mathcal{O}\Big(\frac{n^2}{\lg^{1/5} n}\Big)$, this simplifies to $\mathcal{O}\Big(\lg^2 \frac{n^2}{m} / \lg\lg n\Big)$. Finally, report queries can be solved via select queries, thus taking time $\mathcal{O}\Big((k+1) \lg^2 \frac{n^2}{m} / \lg\lg n\Big)$.

Let us consider the case of rectangular matrices of $n_r \times n_c$. We obtain essentially the same results by using $s = \frac{n_r n_c \lg m}{m}$ and $b = \frac{n_r n_c \lg\lg m}{m}$, so that any time complexity $\lg \frac{n^2}{m}$ becomes $\lg \frac{n_r n_c}{m}$, and any $\lg\lg n$ divisors in time complexities become $\lg\lg(n_r n_c)$. When $m = o(\max(n_r, n_c) \lg(n_r n_c))$ we cannot use those superblock sizes anymore, but we can use just one wavelet tree, which maintains the space and time complexities as before.

6. A Fully Compressed Representation

We now have all the necessary pieces to prove our main result.

Theorem 3. *An $n \times n$ grid with m points can be represented within $H + o(H)$ bits of space, where $H = \lg \binom{n^2}{m}$, so that range counting takes $\mathcal{O}(\lg n / \lg\lg n)$ time, range reporting of k points requires time $\mathcal{O}\big((k+1) \lg^2 n / \lg\lg n\big)$, and point selection queries are solved in $\mathcal{O}\big(\lg^2 n / \lg\lg n\big)$ time.*

Proof. We use our "almost fully compressed" solution (Theorem 2) when $m = \mathcal{O}\left(\frac{n^2}{\lg^{1/5} n}\right)$, and our "compressed" solution (Theorem 1) when $m = \omega\left(\frac{n^2}{\lg^{1/5} n}\right)$ and $n^2 - m = \omega\left(\frac{n^2}{\lg^{1/5} n}\right)$. As shown at the end of Section 5.4, we obtain $H + o(H)$ bits in the first case. In the second case, because of the range of m values we consider, the redundancy is $\mathcal{O}\left(\frac{n^2 \lg\lg n}{\lg^{1/4} n}\right) = o(m)$, and $H \geq m \lg \frac{n^2}{m} \geq m$ for $m \leq n^2/2$. If $m \geq n^2/2$, we have that the redundancy is also $\mathcal{O}\left(\frac{n^2 \lg\lg n}{\lg^{1/4} n}\right) = o(n^2 - m)$, and $H \geq (n^2 - m) \lg \frac{n^2}{n^2 - m} \geq n^2 - m$. Therefore the redundancy is always $o(H)$.

We have not yet handled the case of large $m = n^2 - \mathcal{O}\left(\frac{n^2}{\lg^{1/4} n}\right)$. In this case, we complement the matrix and use Theorem 2 with queries on 0s instead of queries on 1s. This worsens some times, as shown in Section 5.5. Moreover, we must assume the smallest possible values on m for query times that improve with m, because we are using the complemented matrix. □

Note that, although we give the worst cases in the theorem, for most values of m the times are indeed lower. Note also that the times of the theorem are valid as well when querying for both 0s and 1s.

7. Higher Dimensions

We now generalize our results of Section 4 to any dimension d. In principle, the only restriction on d is that the RAM machine can handle coordinates up to $\lg(n^d)$ in constant time, that is, $w = \Omega(d \lg n)$. Our space usage, however, will become $\Omega(n^d)$ bits for relatively small d. Therefore, the d values of interest will be constant or very slightly superconstant.

In the literature d is generally considered constant. Multidimensional rank can be carried out using $\mathcal{O}\big(m(\lg m/\lg\lg m)^{d-2}\big)$ words of space and $\mathcal{O}\big((\lg m/\lg\lg m)^{d-1}\big)$ time [23]. For reporting k points, the best current solutions require $\mathcal{O}\big(m(\lg m/\lg\lg m)^{d-3}\big)$ words of space and $\mathcal{O}\big((\lg m/\lg\lg m)^{d-2} + k\big)$ time [23], or $\mathcal{O}\big(m \lg^{d-2+\epsilon} m\big)$ words of space and $\mathcal{O}\big((\lg m/\lg\lg m)^{d-3} \lg\lg m + k\big)$ time [7]. When the lower coordinate of the query in every dimension is 1, one can obtain $\mathcal{O}\big(m(\lg m/\lg\lg m)^{d-3}\big)$ words of space and $\mathcal{O}\big((\lg m/\lg\lg m)^{d-3} \lg\lg m + k\big)$ time [8]. Chazelle [10] proved that any reporting time of the form $\mathcal{O}(\text{polylog}(m) + k)$ requires $\Omega(m(\lg m/\lg\lg m)^{d-1})$ words of space on a pointer machine (note that the upper bounds, which are on the RAM model, slightly break this lower bound).

Again, we obtain better times by using space $H + o(n^d)$, where now $H = \lg \binom{n^d}{m}$. This space is $H + o(H)$ when the matrix is sufficiently dense, as before. We first consider rank and select queries, which are relatively simple, then report queries in 3 dimensions and, finally, report queries for $d \geq 4$.

7.1. Rank and Select Queries

It is not difficult to generalize our bounds for rank. To do this, we subdivide the matrix into superblocks of size s^d with each edge of length $s = \lg^2 n$. We further subdivide the superblocks into blocks of size b^d with each edge of length $b = \sqrt[d]{\frac{\lg n}{2}}$. As before, we store the rank values at the sides of the superblocks and the local rank values at the sides of the blocks.

We use the same encoding technique of Section 4.4. It is not hard to see that the space adds up to $H + \mathcal{O}\Big(\frac{n^d \lg(b^d)}{b^d}\Big) = H + \mathcal{O}\Big(\frac{n^d \lg\lg n}{\lg n}\Big) = H + o(n^d)$. To this we must add the space required by the rank values stored along the sides of all the s^d-sized cubes. Since d-dimensional cubes have $2d$ faces, each containing s^{d-1} cells, and we must store numbers up to n^d, we require in total $\mathcal{O}\Big(\frac{n^d}{s^d}\ ds^{d-1}\ \lg(n^d)\Big) = \mathcal{O}\Big(\frac{d^2 n^d}{\lg n}\Big)$ bits for the superblock counters. Similarly, for blocks, which require only $\lg(s^d)$ bits per counter, the space required is $\mathcal{O}\Big(\frac{n^d}{b^d}\ db^{d-1}\ \lg(s^d)\Big) = \mathcal{O}\Big(\frac{d^2 n^d \lg\lg n}{\lg^{1/d} n}\Big)$. Finally, we must precompute answers to every possible dominance query inside every possible block, which adds up to $\mathcal{O}\Big(b^d \cdot 2^{b^d} \cdot \lg(b^d)\Big) = \mathcal{O}(\sqrt{n} \lg n \lg\lg n)$ bits. Therefore the total space can be written as $H + \mathcal{O}\Big(\frac{d^2 n^d \lg\lg n}{\lg^{1/d} n}\Big)$, which is $H + o(n^d)$ for $d \leq \lg\lg n/(3 \lg\lg\lg n)$.

The computation of rank takes constant time for constant d, by adding and subtracting counters at the sides of superblocks and blocks, and finally using the universal tables over the $b^d = \frac{\lg n}{2}$ bits of a b^d-sized cube to complete the computation. To compute the time as a function of d, consider that we have to project the point into every face of the superblock (i.e., we project one dimension, in $\binom{d}{1}$ ways). This counts more than once those points covered when projecting any two dimensions, so we must project two dimensions, to subtract those ranks, in $\binom{d}{2}$ ways. Points covered by three projections have been subtracted more than once, then we have to project three dimensions, in $\binom{d}{3}$ ways, to add back those ranks, and so on. All those combinatorials add up to 2^d computations we must perform. Thus the time for rank is $\mathcal{O}\big(2^d\big)$.

We can also easily generalize our bounds for select. Suppose we are given an ordering on the dimensions and asked to find the kth point in an arbitrary

d-dimensional box. We can again use binary search with rank to find that point in $\mathcal{O}(\lg n)$ time, assuming d is constant. The cost as a function of n and d is $\mathcal{O}\big(2^d \lg(n^d)\big) = \mathcal{O}\big(2^d d \lg n\big)$.

Now suppose one corner of the box is at the origin, that is, the point whose coordinates are all 1. We start by considering only the first dimension. For each beam of cells running along that dimension, we pick one value out of s, and store all the rank values at those points in a y-fast trie data structure. This requires $\mathcal{O}\big(n^{d-1}\frac{n}{s}\lg(n^d)\big) = \mathcal{O}\Big(\frac{dn^d}{\lg n}\Big)$ extra bits. This trie tells us, in $\mathcal{O}(\lg\lg n)$ time, the interval of length s where the select answer lies along a chosen beam. Then a binary search using rank finds the precise point in time $\mathcal{O}\big(2^d \lg s\big) = \mathcal{O}\big(2^d \lg\lg n\big)$. This point determines the $(d-1)$-dimensional plane, perpendicular to the first dimension, that contains the kth point in the box. This leaves us with the same problem in $d-1$ dimensions; when we reach 2 dimensions, we can apply Theorem 1. The space cost of the solution for the d dimensions is $\mathcal{O}\Big(\frac{d^2 n^d}{\lg n}\Big)$ bits, and the time is $\mathcal{O}\big(2^d d \lg\lg n\big)$. This time is $\mathcal{O}(\lg\lg n)$ for constant d.

Finally, note that we may wish to choose any order of dimensions of select at query time. In such a case the space overheads have to be added for each ordering of the dimensions, multiplying the space by a factor of $d!$. This yields space $\mathcal{O}\Big(\frac{d^2 d! n^d}{\lg n}\Big)$, which is $o(n^d)$ for $d \le \lg\lg n / \lg\lg\lg n$.

7.2. Report Queries in 3 Dimensions

It is not so easy to generalize report, the main difficulty being to generalize *nextCol* to *nextPlane*, which takes a range in each of the first $d-1$ dimensions and a starting point j in the last dimension, and returns the smallest $j' > j$ such that there is a point contained in the intersection of those ranges and the plane whose last coordinate is j'. To apply the technique of Section 4.2, we would need an extension of the one-dimensional RMQ techniques we used to multidimensional arrays. Although there is a linear-bits space index solving RMQs in constant time for two dimensions [12] (which needs access to the array, however), the best result for higher dimensions [34] takes linear-words space, which is too high for us. In addition, the simple division into chunks and within-chunk bands becomes much more complicated in two and more dimensions, as we see soon. Therefore, we develop a different technique, akin to a classical idea used for one-dimensional RMQs in the past [4, 5] (i.e., to consider any one-dimensional range as the union of at most two ranges whose sizes are powers of 2).

Let us first consider the relatively simple case $d = 3$. We take the 2-dimensional submatrix obtained by ignoring the last dimension, and divide it into 2-dimensional batches of size $s \times s$, where again $s = \lg^2 n$.

For each rectangle of batches whose side lengths are both powers of 2 when measured in batches, we store a signature bit vector indicating for which coordinates in the third dimension that rectangle contains at least one point. We call such rectangles *type 1 rectangles*, and their signature bit vectors take a total of $\mathcal{O}\Big(n\left(\frac{n}{s}\right)^2 \lg^2 n\Big) = o(n^3)$ bits. We also store a signature bit vector for each rectangular range $[s\cdot(i_s-1)+1, s\cdot(i_s+2^{k_1}-1)]\times[j, j+2^{k_2}-1]$ with $1 \le i_s \le n/s$, $1 \le j \le n$, $0 \le k_1 \le \lg(n/s)$ and $0 \le k_2 \le \lg s$, and similarly in the other direction, $[i, i+2^{k_2}-1]\times[s\cdot(j_s-1)+1, s\cdot(j_s+2^{k_1}-1)]$ for $1 \le i \le n$ and $1 \le j_s \le n/s$. We call such rectangular ranges *type 2 rectangles*, and their signature bit vectors take a total of $\mathcal{O}\big(n\left(\frac{n}{s}\right) n \lg n \lg s\big) = o(n^3)$ bits (this space dominates that of type 1 rectangles). Notice that any 2-dimensional rectangular range can be expressed as the union of at most 4 type 1 rectangles, at most 16 type 2 rectangles (4 at each of the 4 sides of the main rectangle), and at most 4 rectangular ranges completely contained within single batches.

We deal with the ranges within single batches by dividing each batch into 2-dimensional chunks of size $r \times r$, where this time $r = \lg^{1/9} n$. We use the same machinery for chunks as we did for batches. The extra space for all the chunk-level bit vectors is $\mathcal{O}\big(n\left(\frac{n}{r}\right) n \lg s \lg r\big) = \mathcal{O}\Big(\frac{n^3(\lg\lg n)^2}{\lg^{1/9} n}\Big) = o(n^3)$ bits. Finally, there are $\mathcal{O}((n/r)^2 r^4) = \mathcal{O}\Big(n^2 \lg^{2/9} n\Big)$ rectangular ranges completely contained within single chunks. Applying Lemma 1, with our b^3-sized matrix blocks and $b = \sqrt[3]{\frac{\lg n}{2}}$, we store indexes for them all in $\mathcal{O}\Big(n^2 \lg^{2/9} n \cdot \frac{n \lg\lg n}{(\lg(n)/2)^{1/3}}\Big) = o(n^3)$ bits. Once we have solved *nextPlane* in one dimension, we solve *nextCol* as usual on the two remaining dimensions. The only difference is that we use this b value instead of the $\sqrt{\frac{\lg n}{2}}$ value used in Section 4, as we already have the b^3-sized cubes encoded for three dimensions.

7.3. Report Queries in Higher Dimensions

There are many more types than 1 and 2 in higher dimensions. Along each dimension (except the one we are eliminating) we can choose to sample in the form $[s\cdot(i_s-1)+1, s\cdot(i_s+2^{k_1}-1)]$ or in the form $[j, j+2^{k_2}-1]$.

The number of bits we use at the batch level can thus be bounded by

$$\begin{aligned}
& \mathcal{O}\left(n \cdot \sum_{1 \le \ell \le d-1} \binom{d-1}{\ell} \left(\frac{n}{s}\right)^{\ell} n^{d-\ell-1} \lg^{\ell} n \lg^{d-\ell-1} s\right) \\
= \; & \mathcal{O}\left(n \cdot \left(\left(\frac{n \lg n}{s} + n \lg s\right)^{d-1} - (n \lg s)^{d-1}\right)\right) \\
= \; & \mathcal{O}\left(n \cdot (d-1) \frac{n \lg n}{s} (n \lg s)^{d-2}\right) \\
= \; & \mathcal{O}\left(\frac{d n^d (\lg \lg n)^{d-2}}{\lg n}\right),
\end{aligned}$$

which is $o(n^d)$ for $d \le \lg \lg n / \lg \lg \lg n$.

We set the side length of each chunk to $r = \lg^{1/d^2} n$. The number of bits we use at the chunk level can be bounded, analogously, by

$$\begin{aligned}
& \mathcal{O}\left(n \cdot \sum_{1 \le \ell \le d-1} \binom{d-1}{\ell} \left(\frac{n}{r}\right)^{\ell} n^{d-\ell-1} \lg^{\ell} s \lg^{d-\ell-1} r\right) \\
= \; & \mathcal{O}\left(n \cdot (d-1) \frac{n \lg s}{r} (n \lg r)^{d-2}\right) \quad = \quad \mathcal{O}\left(\frac{n^d (\lg \lg n)^{d-1}}{d^{2d-5} \lg^{1/d^2} n}\right),
\end{aligned}$$

which is $o(n^d)$ for $d < (3 \lg \lg n / \lg \lg \lg n)^{1/3}$.

Finally, there are $\mathcal{O}\left(\left(\frac{n}{r}\right)^{d-1} r^{2(d-1)}\right)$ rectangular ranges completely contained within single chunks, so applying Lemma 1 with $b = \sqrt[d]{\frac{\lg n}{2}}$, we store indexes for them all in

$$\mathcal{O}\left(\left(\frac{n}{r}\right)^{d-1} r^{2(d-1)} \cdot \frac{n \lg \lg n}{b}\right) = \mathcal{O}\left(\frac{n^d \lg \lg n}{\lg^{1/d^2} n}\right)$$

bits. This is $o(n^d)$ for $d = o(\sqrt{\lg \lg n / \lg \lg \lg n})$. The number of operations is proportional to the number of rectangles needed to cover the query. At the batch level, this is 6^{d-1}, as we show in the Appendix. At the chunk level we have to consider that there are 2^{d-1} vertices in the query rectangle, each of which is covered with chunks using the same technique. This rises the total complexity to $\mathcal{O}(12^d)$.

Once we find the $(d-1)$-dimensional hyperplane where the next point lies, we have the same problem on $d-1$ dimensions. We proceed recursively,

yet we maintain the r and b values we have used for the first problem (as we have the data encoded according to them). The time complexities do not grow when adding over lower dimensions as they are already exponential in d. This gives us our higher-dimensional generalization of Theorem 1.

Theorem 4. *Consider a binary matrix with edge length n in d dimensions, on a RAM machine with words of $\Omega(d \lg n)$ bits. Suppose the matrix contains m 1s and thus has entropy $H = \lg \binom{n^d}{m}$. Then it can be represented within $H + \mathcal{O}\left(\frac{n^d (\lg\lg n)^{d-1}}{d^{2d-5} \lg^{1/d^2} n}\right)$ bits, which is $H + o(n^d)$ for $d < (3 \lg\lg n / \lg\lg\lg n)^{1/3}$, such that* rank *takes $\mathcal{O}\big(2^d\big)$ time (i.e., $\mathcal{O}(1)$ for constant d),* report *takes $\mathcal{O}\big((k+1)12^d\big)$ time (i.e., $\mathcal{O}(k+1)$ for constant d), and* select *takes $\mathcal{O}\big(2^d d \lg n\big)$ time in general or $\mathcal{O}\big(2^d d \lg\lg n\big)$ time when one corner is at the origin (i.e., $\mathcal{O}(\lg n)$ and $\mathcal{O}(\lg\lg n)$ for constant d).*

8. Conclusions

Although orthogonal range searching has received much attention, the interesting case where the structure achieves entropy-bounded space has remained largely under-explored. This work completes a relevant portion of the picture, not only reaching fully-compressed space but also uncovering interesting relations between the time complexities that can be achieved and the entropy of the grid: We show that grids with higher entropy can be queried faster within fully-compressed space. Previous work has been blind to this relation, focusing only on the relation between time and space in terms of the number of points of the grid.

A natural question is what the lower bounds are when the entropy comes into play. While this issue has been studied in the one-dimensional case [15–19, 21, 30, 31], it is new in two and more dimensions (there are some lower bounds on binary matrices, but considering one-dimensional operations like rank and select on rows or columns [16]). The fact that we have presented various upper bounds that are patched to form a complete solution, where some patches have incomparable complexities and there are some abrupt transitions from one patch to another (e.g., see Table 1 and the proof of Theorem 3), suggest that there must be a more uniform complexity as a function of the entropy we have not yet reached.

The relation with the one-dimensional case is also intriguing in some aspects. For example, in one-dimensional bitmaps operation select can be

solved in constant time, and it is actually easier on sparser bitmaps, whereas in more dimensions we have not achieved that. For rank, on the other hand, we have achieved constant time on dense matrices and entropy-related time on sparse ones, much as in some one-dimensional solutions [26].

As for the space, $H = \lg \binom{n^2}{m}$ is a crude worst-case lower bound that does not account for regularities, such as clusters of points, that arise in real life. Our actual space for storing the bitmaps can indeed be much better than H when such regularities arise: It is the sum of local entropies of small blocks. Our $o(H)$-bits indexes on the data, however, reach this space by design and do not improve if the data has regularities. An interesting challenge is to make the redundancy sensitive to the data distribution as well.

Other natural directions for future work would be to consider further operations [3], achieving dynamic compressed structures, and secondary-memory variants. Finally, note that we have extended to d dimensions the variant that uses $H + o(n^d)$ bits, but the strongly 2-dimensional nature of wavelet trees prevented us extending the $H + o(H)$ bit structures analogously.

Acknowledgements. We thank Jérémy Barbay and Yakov Nekrich for discussions, and Meg Gagie for righting our grammar. We also thank the exhaustive work of the reviewers, which decisively helped to improve the presentation.

References

[1] Agarwal, P., Erickson, J., 1999. Geometric range searching and its relatives. In: Advances in Discrete and Computational Geometry. Contemporary Mathematics 223. AMS Press, pp. 1–56.

[2] Alstrup, S., Brodal, G., Rauhe, T., 2000. New data structures for orthogonal range searching. In: Proc. 41st Annual Symposium on Foundations of Computer Science (FOCS). pp. 198–207.

[3] Barbay, J., Claude, F., Navarro, G., 2010. Compact rich-functional binary relation representations. In: Proc. 9th Latin American Symposium on Theoretical Informatics (LATIN). LNCS 6034. pp. 172–185, extended version in `http://arxiv.org/abs/1201.3602`.

[4] Bender, M., Farach-Colton, M., 2000. The LCA problem revisited. In: Proc. 4th Latin American Symposium on Theoretical Informatics (LATIN). pp. 88–94.

[5] Berkman, O., Vishkin, U., 1993. Recursive star-tree parallel data structure. SIAM Journal of Computing 22 (2), 221–242.

[6] Bose, P., He, M., Maheshwari, A., Morin, P., 2009. Succinct orthogonal range search structures on a grid with applications to text indexing. In: Proc. 11th International Symposium on Algorithms and Data Structures (WADS). pp. 98–109.

[7] Chan, T., Larsen, K. G., Pătraşcu, M., 2011. Orthogonal range searching on the RAM, revisited. In: Proc. 27th ACM Symposium on Computational Geometry (SoCG). pp. 1–10, extended version in `http://arxiv.org/abs/1103.5510`.

[8] Chan, T., Pătraşcu, M., 2010. Counting inversions, offline orthogonal range counting, and related problems. In: Proc. 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 161–173.

[9] Chazelle, B., 1986. Filtering search: A new approach to query-answering. SIAM Journal on Computing 15, 703–724.

[10] Chazelle, B., 1990. Lower bounds for orthogonal range searching: I. The reporting case. Journal of the ACM 37 (3), 200–212.

[11] Clark, D., 1996. Compact PAT trees. Ph.D. thesis, University of Waterloo, Canada.

[12] Davoodi, P., 2011. Data structures: Range queries and space efficiency. Ph.D. thesis, Aarhus University, Denmark.

[13] Farzan, A., Gagie, T., Navarro, G., 2010. Entropy-bounded representation of point grids. In: Proc. 21st Annual International Symposium on Algorithms and Computation (ISAAC). LNCS 6507. pp. 327–338 (part II).

[14] Fischer, J., Heun, V., 2011. Space-efficient preprocessing schemes for range minimum queries on static arrays. SIAM Journal of Computing 40 (2), 465–492.

[15] Golynski, A., 2007. Optimal lower bounds for rank and select indexes. Theoretical Computer Science 387 (3), 348–359.

[16] Golynski, A., 2009. Cell probe lower bounds for succinct data structures. In: Proc. 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 625–634.

[17] Golynski, A., Grossi, R., Gupta, A., Raman, R., Rao, S. S., 2007. On the size of succinct indices. In: Proc. 15th Annual European Symposium on Algorithms (ESA). LNCS 4698. pp. 371–382.

[18] Golynski, A., Orlandi, A., Raman, R., Rao, S. S., 2011. Optimal indexes for sparse bit vectors. CoRR abs/1108.2157.

[19] Golynski, A., Raman, R., Rao, S. S., 2008. On the redundancy of succinct data structures. In: Proc. 11th Scandinavian Workshop on Algorithm Theory (SWAT). LNCS 5124. pp. 148–159.

[20] Grossi, R., Gupta, A., Vitter, J., 2003. High-order entropy-compressed text indexes. In: Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 841–850.

[21] Grossi, R., Orlandi, A., Raman, R., Rao, S. S., 2009. More haste, less waste: Lowering the redundancy in fully indexable dictionaries. In: Proc. 26th International Symposium on Theoretical Aspects of Computer Science (STACS). pp. 517–528.

[22] Gupta, A., Hon, W.-K., Shah, R., Vitter, J., 2007. Compressed data structures: Dictionaries and data-aware measures. Theoretical Computer Science 387 (3), 313–331.

[23] JáJá, J., Mortensen, C., Shi, Q., 2004. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In: Proc. 15th International Symposium on Algorithms and Computation (ISAAC). pp. 558–568.

[24] Munro, I., 1996. Tables. In: Proc. 16th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS). LNCS 1180. pp. 37–42.

[25] Nekrich, Y., 2005. Space efficient dynamic orthogonal range reporting. In: Proc. 21st Annual Symposium on Computational Geometry (SoCG). pp. 306–313.

[26] Okanohara, D., Sadakane, K., 2007. Practical entropy-compressed rank/select dictionary. In: Proc. 9th Workshop on Algorithm Engineering and Experiments (ALENEX). pp. 60–70.

[27] Pagh, R., 2001. Low redundancy in static dictionaries with constant query time. SIAM Journal of Computing 31 (2), 353–363.

[28] Pătraşcu, M., 2007. Lower bounds for 2-dimensional range counting. In: Proc. 39th Annual ACM Symposium on Theory of Computing (STOC). pp. 40–46.

[29] Pătraşcu, M., 2008. Succincter. In: Proc. 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS). pp. 305–313.

[30] Pătraşcu, M., Thorup, M., 2006. Time-space trade-offs for predecessor search. In: Proc. 38th Annual ACM Symposium on Theory of Computing (STOC). pp. 232–240.

[31] Pătraşcu, M., Viola, E., 2010. Cell-probe lower bounds for succinct partial sums. In: Proc. 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 117–122.

[32] Raman, R., Raman, V., Rao, S. S., 2007. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. ACM Transactions on Algorithms 3 (4), article 43.

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

[34] Yuan, H., Atallah, M., 2010. Data structures for range minimum queries in multidimensional arrays. In: Proc. 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 150–160.

Appendix A. Number of Rectangles Covering a Batch

Assume the box crosses a batch boundary in some dimension. If it does not, then we handle it at the chunk level, and the calculations are similar. Consider the $(d-1)$-dimensional box as the product of $d-1$ intervals, one in each dimension. Each interval consists of (a) a part of length $< s$ "to the left of" the first complete batch (i.e., whose coordinate along that dimension

is smaller than the minimum value of the first batch), (b) a maximal part consisting of complete batches, and (c) a part of length $< s$ "to the right of" the last complete batch. The last two parts can be empty.

Let us first break the rectangle up into zones, where two points are in the same zone if they are in the same part of each interval. We can count the number of zones as follows: (1) we sum over ℓ from 0 to $d-1$, where ℓ is the number of dimensions for which the zone is in the part consisting of complete batches; (2) there are $\binom{d-1}{d-1-\ell} = \binom{d-1}{\ell}$ ways to choose for which of the remaining dimensions the zone is to the left or right; (3) there are $2^{d-1-\ell}$ ways to choose for which of those remaining dimensions the zone is to the left and for which it is to the right.

Any zone is a $(d-1)$-dimensional box that is aligned with a batch boundary on at least one side and, in every dimension, it is either aligned with batch boundaries on both sides or it is narrower than a batch. Therefore, we can cover it with 2^{d-1} boxes for which we have the vectors stored.

Thus the total number of rectangles we need is at most

$$2^{d-1} \cdot \sum_{\ell=1}^{d-1} \binom{d-1}{\ell} 2^{d-1-\ell} \quad \leq \quad 2^{d-1} \cdot 3^{d-1} \quad = \quad 6^{d-1}.$$