Entropy-Bounded Representation of Point Grids

Arash Farzan[1], Travis Gagie[2,⋆], and Gonzalo Navarro[2,⋆]

[1] Max-Planck-Institut für Informatik
afarzan@mpi-inf.mpg.de
[2] Department of Computer Science, University of Chile
{tgagie, 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 a performance that is comparable to that of uncompressed structures and that improves upon the only previous compressed structure. Operating within entropy-bounded space opens a new line of research on an otherwise well-studied area, and is becoming extremely important for handling large datasets.

1 Introduction

A point grid is an extremely 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 queries are possible if points have associated values, and also more general shapes than rectangles have been considered.

Consider a $n \times n$ grid containing m points. 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, $\mathcal{O}(m)$ integers [12]. The preprocessing time is $\mathcal{O}\left(m\sqrt{\lg m}\right)$ [5]. That counting time cannot be improved within space $\mathcal{O}(m \operatorname{polylog}(m))$ [17]. Range reporting can be done in time $\mathcal{O}(\lg \lg m + k)$, where k is the number of points reported, using $\mathcal{O}(m \lg^{\epsilon} m)$ integers for any constant $\epsilon > 0$ [1]. The time raises to $\mathcal{O}(\lg \lg m(\lg \lg m + k))$ if the space is reduced to $\mathcal{O}(m \lg \lg m)$ integers [1], and it reaches $\mathcal{O}\left(\frac{\lg m}{\lg \lg m} + k \lg^{\epsilon} m\right)$ if the space is linear [15]. There are also some bounds that may be relevant when many points are to be reported: $\mathcal{O}(\lg m + k \lg \lg(4m/k))$ time using $\mathcal{O}(m \lg \lg m)$ integers, and $\mathcal{O}(\lg m + k \lg^{\epsilon}(2m/k))$ time using $\mathcal{O}(m)$ integers [6]. Some of these results have been matched even in the dynamic scenario [14].

Many of the application areas for this problem handle huge volumes of information, and thus even the "linear" space structures might be excessively large.

⋆ Partially funded by Fondecyt Grant 1-080019, Chile.

Since each point requires storing two coordinates, the space complexities above should be multiplied by $2\lg n$ bits. Unless m is very small, even storing these bare coordinates uses much more space than necessary, and moreover the constants hidden within the $\mathcal{O}(...)$ notation are not negligible. Some succinct data structures have been designed using, for example, $m\lg m + o(m\lg m)$ bits when $m = n$, answering range counting queries in time $\mathcal{O}\left(\frac{\lg m}{\lg\lg m}\right)$ and reporting in time $\mathcal{O}\left((k+1)\frac{\lg m}{\lg\lg m}\right)$ [4]. That space is not the best possible when m is larger than 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" $H = \lg\binom{n^2}{m} = m\lg\frac{n^2}{m} + \mathcal{O}(m+\lg n)$.

To the best of our knowledge, the only previous work achieving this "compressed" space is by Barbay et al. [2]. They propose a data structure using $H + o(H) + \mathcal{O}(m) + o(n)$ bits (see our Thm. 2). Within this space they solve many interesting range counting, range reporting, and point selection queries (give the kth point in a rectangle, according to some order) in $\mathcal{O}(\lg n)$ time per delivered point. Contrarily to succinct indexes, they propose an integrated encoding, where the data and the index are stored together.

In this paper we push further in the direction of storing the grid data within its entropy bound, improving simultaneously the space redundancy and the time performance of queries. Most notably, we achieve a *fully compressed* representation taking $H + o(H)$ bits of space, while supporting the operations in constant time in some cases. Depending on m, we use different data structures to achieve this goal. The result is summarized in Thm. 1; see Section 2.2 for more details.

Theorem 1. *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 orthogonal range counting can be answered in $\mathcal{O}\left(\lg\frac{n^2}{m}\right)$ time, range reporting in time $\mathcal{O}\left(\lg^2\frac{n^2}{m}\right)$ per delivered datum, and point selection queries in at most $\mathcal{O}\left(\lg^2 n\right)$ time. Depending on the density of the matrix the times are reduced down to $\mathcal{O}(1)$ per delivered datum.*

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 describes a "compressed" representation taking $H + o(n^2)$ bits of space and achieving constant time for range counting and reporting. Section 4 achieves the "fully compressed" space, in exchange for higher query times, and finishes with the proof of Thm. 1. Section 5 concludes and gives further research directions.

2 Basic Concepts

2.1 The One-Dimensional Case

The one-dimensional variant of the problem, i.e., on a bitmap $B[1,n]$, has been long studied. Let B have m bits set, then the entropy of the bitmap is $H = \lg\binom{n}{m}$.

All the range counting, range reporting, and point locating queries can be solved in terms of two primitives: $\mathsf{rank}_b(B,i)$ is the number of occurrences of bit b in $B[1,i]$, and $\mathsf{select}_b(B,j)$ is the position in B of the jth occurrence of bit b.

Clark [8] and Munro [13] showed that both rank and select can be solved in constant time using $n+o(n)$ bits of space, that is, B itself plus sublinear space. Golynski et al. [9] showed that the $o(n)$ term must be $\Omega(\frac{n \lg\lg n}{\lg n})$ if B is stored in plain form, and moreover achieved this bound. Raman et al. [19] provided a compressed representation retaining constant query times and taking $H+\mathcal{O}\Big(m+\frac{n\lg\lg n}{\lg n}\Big)$ bits. We prove now a technical lemma we will need later.

Lemma 1. *Let $0<\alpha\le 1$ be a constant and $b=\Theta(\lg^\alpha n)$. Let bitmap $B[1,n]$ be stored in a way such that we can only access pieces $B[(i-1)\cdot b+1, i\cdot b]$ at a 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.*

Proof. Let us take any algorithm achieving constant time and $\mathcal{O}\Big(\frac{n\lg\lg n}{\lg n}\Big)$ extra space, say Golynski's [9], and adapt it to this restriction. The algorithm builds and uses several indexes and accesses B a constant number of times. Each such time, it reads a "word" of $w=\mathcal{O}(\lg n)$ 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.

We introduce an indirection when accessing such universal tables. Each word is covered by $\frac{w}{b}$ pieces. For each piece, we store the *summary* number of 1s in the piece. 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 pieces 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 size $\mathcal{O}\big(2^{o(\lg n)}\mathrm{polylog}(n)\big)$, telling (a) the number of bits set up to any given piece of the word, and (b) the piece where the kth 0/1 of the word occurs. A final access to one b-bit piece, with another universal table of size $\mathcal{O}\big(2^b\mathrm{polylog}(n)\big)$, completes the query in constant time.

The lower bound comes directly from Golynski [9], who states that if one probes t bits and answers $\mathsf{rank}/\mathsf{select}$ in constant time, then the index must be of size $\Omega(\frac{m\lg t}{t})$. In the worst case $m=n$ and the algorithm can access at most $t=\mathcal{O}(b)$ bits in constant time. □

2.2 Two Dimensions

We can identify a grid with a binary matrix. We will consider rectangular ranges of the form $(i_1,i_2)\times(j_1,j_2)$, where i_1 and i_2 are rows and j_1 and j_2 are columns. 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 row-major or column-major order (this generalizes from 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 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. 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 (up to) 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 within the time of selecting the k_1th point and then reporting the $k_2 - k_1 + 1$ points following it. Furthermore, our rank query is never slower than our select, and $\mathsf{select}(i_1, i_2, j_1, j_2, k) = \mathsf{select}(i_1, i_2, 1, n, k + x)$ with $x = \mathsf{rank}(i_1, i_2, 1, j_1 - 1)$ if select delivers in column-major order, and analogously in row-major order. 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 (as explained this allows one to emulate any 4-sided query);
- $\mathsf{select}(i, k)$ gives the kth point in the range $(1, i) \times (1, n)$, in column-major order (this allows one to emulate any 3-sided query); and
- $\mathsf{report}(i_1, i_2, j_1, k)$ gives the first (up to) k points in the range $(i_1, i_2) \times (j_1, n)$, in column-major order.

Barbay et al. [2] propose a number of primitives on binary matrices, yet several can be reduced to others. Their maximal operations (in the sense that the others reduce to a constant number of applications of these) are `rel_rnk` (equivalent to our rank), `rel_sel_obj_maj` and `rel_sel_lab_maj` (equivalent to our select), and `lab_rnk` and `obj_rnk` (which count the number of nonempty rows/columns within a range and have no equivalent in this paper). By using *wavelet trees* [10], they achieve the following result (adapted and fixed here):

Theorem 2 ([2]). *A binary matrix of σ rows ("labels") by n columns ("objects") with t 1s can be represented within $H + o(H) + \mathcal{O}\left(t + \frac{n \lg \lg n}{\lg n}\right)$ bits, so that queries* `rel_rnk`(i_1, i_2, j_1, j_2) *(number of points in $(i_1, i_2) \times (j_1, j_2)$),* `rel_sel_lab_maj`$(i, k, j_1, j_2)$ *(kth point, in label-major order, in $(i, \sigma) \times (j_1, j_2)$), and* `rel_min_obj_maj`$(i_1, i_2, j)$ *and* `rel_acc_obj_maj`(i_1, i_2, j) *(first and successive points, in object-major order, in $(i_1, i_2) \times (j, n)$), can be answered in $\mathcal{O}(\lg \sigma)$ time per delivered datum.* `Rel_sel_obj_maj`(i_1, i_2, j, k) *(kth point, in object-major order, in $(i_1, i_2) \times (j, n)$), can be carried out in $\mathcal{O}(\lg \sigma \lg n)$ time.*

Table 1. Space and time complexities achieved by Barbay et al. [2] and in this paper. The "or" case depends on using row-major or column-major order. The times for Thm. 4 are simplified assuming $m = \mathcal{O}\left(\frac{n^2}{\lg^{1/4} n}\right)$; otherwise Thm. 3 takes over. The times to operate on 0s are the same for Thm. 3; for Thm. 4 we give them explicitly.

Source	Space	rank time	report time
Thm. 2 [2]	$H + o(H) + \mathcal{O}(m) + o(n)$	$\lg n$	$(k+1)\lg n$
Thm. 3	$H + \mathcal{O}\left(\frac{n^2 \lg\lg n}{\lg^{1/4} n}\right)$	1	$k+1$
Thm. 4	$H + o(H) + \mathcal{O}(m + \lg n)$	$\lg \frac{n^2}{m}$	$(k+1)\lg \frac{n^2}{m}$
Thm. 4 (0s)		$\lg \frac{n^2}{m}$	$(k+1)\lg^2 \frac{n^2}{m}$
Thm. 1	$H + o(H)$	$\lg \frac{n^2}{m}$	$(k+1)\lg^2 n$

Source	select time (4-sided)	select time (3-sided)
Thm. 2 [2]	$\lg n$ or $\lg^2 n$	$\lg n$
Thm. 3	$\lg n$	$\lg\lg n$
Thm. 4	$\lg n$ or $\lg n + \lg^2 \frac{n^2}{m}$	$\lg \frac{n^2}{m}$ or $\lg n + \lg^2 \frac{n^2}{m}$
Thm. 4 (0s)	$\lg n + \lg^2 \frac{n^2}{m}$	$\lg n + \lg^2 \frac{n^2}{m}$
Thm. 1	$\lg^2 n$	$\lg^2 n$

Proof. This is in their Thm. 2 [2]. Their space formula "$t \lg \sigma + o(t) \lg \sigma + \mathcal{O}\big(\min(t, n \lg \frac{t}{n})\big)$" should indeed be $t \lg \sigma + o(t) \lg \sigma + \lg \binom{n+t}{t} + \mathcal{O}(\min(n,t)) + \mathcal{O}\left(\frac{(n+t)\lg\lg(n+t)}{\lg(n+t)}\right)$. Since the last three terms are $t \lg \frac{n}{t} + \mathcal{O}\left(t + \frac{n \lg\lg n}{\lg n}\right)$, we have the total $t \lg \frac{n\sigma}{t} + o(t) \lg \frac{n\sigma}{t} + \mathcal{O}\left(t + \frac{n\lg\lg n}{\lg n}\right) = H + o(H) + \mathcal{O}\left(t + \frac{n \lg\lg n}{\lg n}\right)$. □

Table 1 compares the previous and new complexities achieved for our operations. The previous compressed representation [2] achieves $H + o(H)$ bits only if $\omega(\frac{n \lg\lg n}{\lg^2 n}) = m = o(n^2)$ (note our m is their t). Also, it always supports rank in time $\mathcal{O}(\lg n)$. This time is $\mathcal{O}(1)$ in our "compressed" solution, and in our "fully compressed" solution it is $\mathcal{O}\left(\lg \frac{n^2}{m}\right)$ in the range $m = \mathcal{O}\left(\frac{n^2}{\lg^{1/4} n}\right)$. This is never worse than the previous result [2], and is strictly better if $m = n^{2-o(1)}$. For report and select we are faster or slower depending on the case.

3 A Compressed Representation

We first describe a solution using $n^2 + o(n^2)$ bits, and then convert it into one using $H + o(n^2)$ bits.

3.1 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 $\mathsf{rank}(i, s \cdot j)$ and $\mathsf{rank}(s \cdot i, j)$ values. 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 is all $\mathsf{rank}_s(i, b \cdot j)$ and $\mathsf{rank}_s(b \cdot i, j)$ values. 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 $0 \le i_{rs}, j_{rs} < s$ 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 $0 \le i_{rb}, j_{rb} < b$ the projections into, and local coordinates within, the blocks. Then it is easy to verify that

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

where $\mathsf{rank}_b(i, j)$ is the local rank value within its block. All the rank and rank_s values in the formula are stored. As for $\mathsf{rank}_b(i, j)$, this is $\mathsf{rank}(i_{rb}, j_{rb})$ within its block. As there are only $2^{b^2} = \sqrt{n}$ different blocks, we can store all the answers to all possible (local) rank 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.

3.2 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, $\mathit{nextCol}(i_1, i_2, j)$ 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 space.

The key idea is to keep signature bit vectors which represent the bitwise-*or* of various contiguous ranges of matrix rows. First we divide the rows into *batches* of $s = \lg^2 n$ rows. Akin to the classical solution to range minimum queries (RMQs) [3], we explicitly store bit vectors of length n which are the *or* of batches i to $i + 2^k - 1$ for all $1 \le i \le n/s$, $0 \le k \le \lg(n/s)$. Furthermore, we enhance these bit vectors with one-dimensional constant-time rank and select structures. This requires $\mathcal{O}\left(\frac{n}{\lg^2 n} \cdot n \lg n\right) = o(n^2)$ bits and reduces the query $\mathit{nextCol}(s \cdot i_1, s \cdot i_2 - 1, j)$ to that of finding the next 1 after position j in either of two bit vectors (the one *or*-ing batches i_1 to $i_1 + 2^k - 1$ and the one *or*-ing batches $i_2 - 2^k$ to $i_2 - 1$, for $k = \lfloor \lg(i_2 - i_1) \rfloor$). Finding the next 1 in a bit vector is easily reduced to one-dimensional rank and select queries, $j' = \mathsf{select}_1(B, \mathsf{rank}_1(B, j) + 1)$.

A general range $[i_1, i_2]$ may contain several batches, plus possibly two within-batch areas at each extreme. Thus we have reduced the problem to within batches of size $\lg^2 n$. We now repeat the partition similarly within each batch. We divide the rows into *chunks* of $d = \lg^{1/4} n$ rows and again use the same machinery to isolate the problem to within chunks. The extra space for all the chunk-level bit vectors is $\mathcal{O}\Big(\frac{n}{\lg^{1/4} n} \cdot n \lg(\lg^2 n)\Big) = o(n^2)$.

Now, confined within a chunk of d rows, we consider bit vectors $B(i_1, i_2)$, $1 \leq i_1, i_2 < d$, such that $B(i_1, i_2)$ is the *or* of rows from i_1 to i_2. We cannot explicitly store all these vectors, as the space would be $\omega(n^2)$. However, we do explicitly 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 $b \times b$ matrix blocks stored contiguously, in order to provide in constant time any $\mathcal{O}\big(\sqrt{\lg n}\big)$ bits of any horizontal strip of width $i_2 - i_1 + 1$. By Lemma 1, we can in this case achieve constant time for rank and select using extra indexes of size $\mathcal{O}\Big(\frac{n \lg\lg n}{\sqrt{\lg n}}\Big)$.

As there are $\mathcal{O}\Big(\frac{n}{\lg^{1/4} n}\Big)$ chunks, each storing $\mathcal{O}\Big((\lg^{1/4} n)^2\Big)$ indexes for $B(i_1, i_2)$, the total space is $\mathcal{O}\Big(\frac{n}{\lg^{1/4} n} \cdot \sqrt{\lg n} \cdot \frac{n \lg\lg n}{\sqrt{\lg n}}\Big) = o(n^2)$ bits. The $nextCol(i_1, i_2, j)$ query is thus solved by consulting at most 2 batch bitmaps, 4 chunk bitmaps, and 2 (virtual) $B(i_1, i_2)$ bitmaps. With rank and select on each, we easily find the next 1 after position j across the 8 bit vectors, in constant time.

Once $nextCol$ is solved, it is easy to address 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 $\mathcal{O}\Big(\frac{n \lg\lg n}{\sqrt{\lg n}}\Big) = o(n)$ per column as we can access only $\mathcal{O}\big(\sqrt{\lg n}\big)$ 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 $\mathcal{O}(k + 1)$.

3.3 *Select* Queries

For 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 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 moreover in a y-fast trie data structure [20]. This sums to $\mathcal{O}\Big(\frac{n \lg n}{\lg^2 n}\Big) = o(n)$ 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 successor of k). Now a binary search over $\lg^2 n$ values gives, in another $\mathcal{O}(\lg\lg n)$ time, the precise column, and one-dimensional rank and select on the column give the position of the kth point. Thus the time is $\mathcal{O}(\lg\lg n)$.

3.4 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 Raman et al. [19] do for one-dimensional bit vectors. Let a block contain m 1s. Then its *class* c is m and its *offset* o is the index of this particular $b \times b$ block among all the different blocks of class m. A table indexed by c and o storing the contents of all the possible bit vectors takes $\mathcal{O}(\sqrt{n}\lg n)$ bits, thus we can recover any block content in constant time.

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 rth block contains m_r bits set, is $\sum_r \lceil \lg \binom{b^2}{m_r} \rceil \leq \lg \binom{n^2}{m} + \mathcal{O}\left(\frac{n^2}{\lg n}\right)$ [19]. Finally, we also need pointers to find in constant time an o field, as these have variable-length representations. This can also be done within $\mathcal{O}\left(\frac{n^2 \lg\lg n}{\lg n}\right)$ bits [19] with techniques akin to one-dimensional rank.

Theorem 3. *A $n \times n$ matrix with m 1s and entropy $H = \lg \binom{n^2}{m}$ can be represented within $H + \mathcal{O}\left(\frac{n^2 \lg\lg n}{\lg^{1/4} n}\right)$ bits, so that operation* $\mathsf{rank}(i,j)$ *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* $\mathsf{select}(i, k)$ *is computed in $\mathcal{O}(\lg\lg n)$ time.*

Note that we can define the complementary queries, where 0s are considered instead of 1s. This is obvious for rank but not for report nor select. It is not hard to see that we can support in addition these complementary queries, by adding other similar $o(n^2)$ bits of space, 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.

4 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. 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. The price will be superconstant time for all queries, due to our internal usage of Thm. 2.

4.1 *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 later in Section 4.4). 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 3.1, 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 $\mathcal{O}(m)$ bits. The latter require $\lg s^2$ bits per datum, adding up to $\mathcal{O}\left(\frac{n^2}{b} \cdot \lg s^2\right) = \mathcal{O}\left(\frac{m}{\lg\lg m} \cdot \lg \frac{n^2 \lg m}{m}\right) = o(H) + \mathcal{O}(m + \lg n)$.

As before, the problem is reduced to supporting local rank within a block of size b^2. We store each block using the wavelet tree of Thm. 2, which for the rth block with m_r bits set requires $\lg\binom{b^2}{m_r} + o(\lg\binom{b^2}{m_r}) + \mathcal{O}(m_r) + o(b)$.[1] It answers rank in time $\mathcal{O}(\lg b) = \mathcal{O}\Big(\lg\frac{n^2}{m} + \lg\lg\lg m\Big)$. Added over all the blocks, the space is $\lg\binom{n^2}{m} + o(\lg\binom{n^2}{m}) + \mathcal{O}(m) + o(m) = H + o(H) + \mathcal{O}(m)$.

4.2 *Select* Queries

As we have stored all the values $\mathsf{rank}(i, s\cdot j)$, $\mathsf{rank}(s\cdot i, j)$, and $\mathsf{rank}_s(i, b\cdot j)$, we can compute any $\mathsf{rank}(i_1, i_2, b\cdot j_1, b\cdot j_2)$ 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 $\mathcal{O}\big(\lg\frac{n}{b}\big)$. For 3-sided queries we can arrange the superblock ranks of each row in a y-fast trie as before, so as to pay $\mathcal{O}(\lg\lg m)$ time to find the superblock, plus $\mathcal{O}\big(\lg\frac{s}{b}\big) = \mathcal{O}(\lg\lg m)$ to binary search for the block.

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 refine the search to find the exact column where the answer lies. A general way to do this is to carry out a binary search within columns $[b\cdot(j_b-1)+1, b\cdot j_b]$ using rank. This rank is not constant-time because we are not in borders of blocks. Hence the time raises to $\mathcal{O}\big(\lg^2 b\big) = \mathcal{O}\Big(\lg^2\frac{n^2}{m} + (\lg\lg\lg m)^2\Big)$. Once we know the precise 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 binary search the block vertically in time $\mathcal{O}\big(\lg\frac{n}{b}\big)$, since we can compute any $\mathsf{rank}(b\cdot i, j)$ value in constant time. Finally, confined within a block, we report the correct point in $\mathcal{O}\big(\lg^2 b\big)$ time using `rel_sel_obj_maj` on the wavelet tree of the block (or $\mathcal{O}(\lg b)$ using `rel_sel_lab_maj`, depending on the orientation).

A smarter way, but one which applies only to one direction (that is, we cannot have simultaneously the improved result for queries $\mathsf{select}(i_1, i_2, 1, n, k)$ and $\mathsf{select}(1, n, j_1, j_2, k)$), is to arrange the block contents in a different way. Instead of using one wavelet tree structure per block, we pack a whole block column per wavelet tree, taking the b columns as their σ "labels" and the n rows as their "objects" (recall Thm. 2). So the rank times are still $\mathcal{O}(\lg\sigma) = \mathcal{O}(\lg b)$, and the within-block rank used in Section 4.1 can still be carried out within this time. Furthermore, finding the k'th point, in label-major order, between objects i_1 and i_2 (operation `rel_sel_lab_maj`), also takes $\mathcal{O}(\lg b)$ time.

Therefore, if the band of our query is horizontal and we have stored block columns in wavelet trees (or vice versa), we find the kth point within time $\mathcal{O}(\lg n)$ (4-sided select) or $\mathcal{O}\Big(\lg\lg m + \lg\frac{n^2}{m}\Big)$ (3-sided select).

4.3 Range Reporting

Let us first assume that the query band is horizontal and we have stored rows of blocks in wavelet trees. To solve $\mathsf{report}(i_1, i_2, j_1, j_2, k)$, we first need to find

[1] The $o(\lg\binom{b^2}{m_r})$ term is asymptotic in b, which is $\omega(1)$ in terms of n, so it can be safely added up over many blocks later.

the next column after $j_1 - 1$ that is nonempty in the range $[i_1, i_2]$. We use the same RMQ-akin idea of Section 3.2. We form batches storing the *or* of ranges of rows of superblocks, for a total space of $\mathcal{O}\big(\frac{n}{s} \cdot n \lg \frac{n}{s}\big) = \mathcal{O}(m)$ bits, and chunks storing the *or* of ranges of rows of blocks within superblocks, for a total space of $\mathcal{O}\big(\frac{n}{b} \cdot n \lg \frac{s}{b}\big) = \mathcal{O}(m)$ bits. Instead of the virtual $B(i_1, i_2)$ bitmaps, to find the next 1 in the band $(i_1, i_2) \times (j+1, n)$ we use the horizontally arranged wavelet trees. They find this point using a `rel_min_obj_maj` query, in $\mathcal{O}(\lg b)$ time.

Now we must be able to find the 1s in the current column j, before proceeding to the next one. Those 1s within the first block since global row i_1 are easily found with `rel_acc_obj_maj` in the wavelet tree of the block. 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 $B_j[i] = 1$ iff there is a 1 in the range $(i \cdot (b-1)+1, i \cdot b) \times (j, j)$ of the matrix. Using one-dimensional rank and select on the B_j vectors, we can easily 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 (unless they exceed the global row i_2). Bit vectors B_j require $\mathcal{O}\Big(\frac{n^2}{b}\Big) = o(m)$ bits of space in total.

If, instead, the band of the query is orthogonal to that of wavelet trees, we proceed as follows. The wavelet tree covering column j_1 can deliver all the points within rows $[i_1, i_2]$ since column j_1 onwards, in $\mathcal{O}(\lg b)$ time each, using `rel_sel_lab_maj`. We must now find the next nonempty block. We build a reduced matrix of $n \times (n/b)$, so that its cell (i', j') contains a 1 iff the original matrix contains a 1 in $(i', b \cdot (j'-1)+1) \times (i', b \cdot j')$. Then the block column that is nonempty in $[i_1, i_2]$ in the original matrix corresponds to the next column that is nonempty in $[i_1, i_2]$ in the reduced matrix. We can then apply the technique of Section 3.2, creating the batch and chunk bitmaps over the reduced matrix, taking overall space $\mathcal{O}\Big(\frac{n^2}{b}\Big) = o(m)$. Once we arrive at the next nonempty block column, its wavelet tree delivers its points in order, and so on.

4.4 The Final Result

A missing piece is to cover the case $m = o(n \lg m) = o(n \lg n)$. When the matrix is so sparse, $\lg \frac{n^2}{m} = \Theta(\lg n)$, and thus it is preferable to use the wavelet tree by itself. The only problem is the $o(n)$ extra space (see Thm. 2), which does not fit in $o(H)$ whenever $m = \mathcal{O}\Big(\frac{n \lg \lg n}{\lg^2 n}\Big)$. This is because Barbay et al. [2] chose Raman et al.'s representation [19] to achieve constant-time rank and select on a bit vector. By using instead a binary searchable representation [11], the $o(n)$ term becomes $o(H) + \mathcal{O}(m + \lg n)$ and the $\mathcal{O}(\lg \sigma)$ time becomes $\mathcal{O}(\lg \sigma + \lg m)$. This is $\mathcal{O}(\lg n)$ for us, which fits perfectly within our general result.

Theorem 4. *A $n \times n$ matrix with m 1s and entropy $H = \lg \binom{n^2}{m}$ can be stored in $H + o(H) + \mathcal{O}(m + \lg n)$ bits, so that operation* $\mathsf{rank}(i, j)$ *is computed in $\mathcal{O}(\tau)$ time and* $\mathsf{report}(i_1, i_2, j_1, j_2, k)$ *in time $\mathcal{O}((k+1)\tau)$, where $\tau = \lg \frac{n^2}{m} + \lg \lg \lg m$. In one direction (that can be chosen),* $\mathsf{select}(i_1, i_2, k)$ *is computed in $\mathcal{O}(\lg n)$*

time, and select(i,k) *in time* $\mathcal{O}(\tau + \lg\lg m)$. *In the other direction, both* select *operations cost* $\mathcal{O}\big(\lg n + \tau^2\big)$ *time.*

Note that if $m = \mathcal{O}\Big(\frac{n^2}{\lg^{1/4} n}\Big)$, then $\tau + \lg\lg m = \mathcal{O}\Big(\lg \frac{n^2}{m}\Big)$; otherwise Thm. 3 is better in all aspects. Finally, if we want to report or select 0s, we can replicate all the extra structures considering the complemented matrix (still defining s and b in terms of 1s), within $o(H) + O(m + \lg n)$ bits . The wavelet trees, instead, must internally binary search on rank to simulate their local operations, all of which (except rank itself) would now cost $\mathcal{O}\big(\tau^2\big)$. Thus this complexity must be added to the times given for report (per delivered datum) and any select.

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

Proof (of Thm. 1). Putting together our "compressed" (Thm. 3) and "fully compressed" (Thm. 4) solutions, the claimed time complexities are obtained. We achieve $H + o(H)$ bit space for all cases: (a) The $\mathcal{O}\Big(\frac{n^2 \lg\lg n}{\lg^{1/4} n}\Big)$ extra bits of our "compressed" solution is $o(H)$ as long as $\omega(\frac{n^2}{\lg^{1/4} n}) = m = n^2 - \omega(\frac{n^2}{\lg^{1/4} n})$. ($b$) The "fully compressed" solution uses $H + o(H) + \mathcal{O}(\lg n)$ space as long as $m = o(n^2)$. (c) These two cover the entire range for m except $m = n^2 - \mathcal{O}\Big(\frac{n}{\lg^{1/4} n}\Big)$; there, we complement the matrix and use the fully-compressed solution with 0-queries instead of 1-queries. (d) The final $\mathcal{O}(\lg n)$ is not $o(H)$ only if $m = \mathcal{O}(1)$, in which case we can encode the points in differential form (*e*.g. δ-encoding), and answer queries in constant time by scanning the points. □

5 Conclusions

Although the area of orthogonal range queries has received much attention, the extremely interesting case where the structure achieves entropy-bounded space is largely under-explored. This work completes a large portion of the picture, and hopefully opens the door to much further research.

A first line of future work is to find and reach the lower bounds on the time complexity under this new scenario. Existing lower bounds, such as $\Omega\left(\frac{\lg m}{\lg \frac{2M}{m}}\right)$ counting time when using M words of memory [7], or one-dimensional lower bounds [18] might be useful. Succinct-space results [4] suggest we can do better. Particularly intriguing is that select takes constant time on dense one-dimensional bitmaps, whereas we have not achieved that in two dimensions. Interestingly, schemes that achieve entropy-bounded extra space in one dimension [16], offer rank time analogous to ours, $\mathcal{O}\big(\lg \frac{n}{m}\big)$, yet their select is constant-time.

As for 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 is indeed much better: the sum of local entropies of small blocks. An interesting future work challenge is to improve the lower order term, $o(H)$.

Other natural directions for future work are to consider further operations [2], extending to d-dimensional spaces, achieving dynamic compressed structures, and secondary-memory variants.

Acknowledgements. We thank Jérémy Barbay for discussions and proofreading.

References

1. Alstrup, S., Brodal, G., Rauhe, T.: New data structures for orthogonal range searching. In: Proc. 41st FOCS, pp. 198–207 (2000)
2. Barbay, J., Claude, F., Navarro, G.: Compact rich-functional binary relation representations. In: López-Ortiz, A. (ed.) LATIN 2010. LNCS, vol. 6034, pp. 172–185. Springer, Heidelberg (2010)
3. Bender, M., Farach-Colton, M.: The level ancestor problem simplified. Theoretical Computer Science 321(1), 5–12 (2004)
4. Bose, P., He, M., Maheshwari, A., Morin, P.: Succinct orthogonal range search structures on a grid with applications to text indexing. In: Proc. 11th WADS, pp. 98–109 (2009)
5. Chan, T., Pătraşcu, M.: Counting inversions, offline orthogonal range counting, and related problems. In: Proc. 21st SODA, pp. 161–173 (2010)
6. Chazelle, B.: Filtering search: A new approach to query-answering. SIAM Journal of Computing 15, 703–724 (1986)
7. Chazelle, B.: Lower bounds for orthogonal range searching: II. The arithmetic model. Journal of the ACM 37(3), 430–463 (1990)
8. Clark, D.: Compact Pat Trees. PhD thesis, University of Waterloo, Canada (1996)
9. Golynski, A.: Optimal lower bounds for rank and select indexes. Theoretical Computer Science 387(3), 348–359 (2007)
10. Grossi, R., Gupta, A., Vitter, J.: High-order entropy-compressed text indexes. In: Proc. 14th SODA (2003)
11. Gupta, A., Hon, W.-K., Shah, R., Vitter, J.S.: Compressed data structures: Dictionaries and data-aware measures. In: Proc. 16th DCC, pp. 213–222 (2006)
12. Já Já, J., Mortensen, C.W., Shi, Q.: Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 558–568. Springer, Heidelberg (2004)
13. Munro, I.: Tables. In: Chandru, V., Vinay, V. (eds.) FSTTCS 1996. LNCS, vol. 1180, pp. 37–42. Springer, Heidelberg (1996)
14. Nekrich, Y.: Space efficient dynamic orthogonal range reporting. In: Proc. 21st SCG, pp. 306–313 (2005)
15. Nekrich, Y.: Orthogonal range searching in linear and almost-linear space. Computational Geometry: Theory and Applications 42(4), 342–351 (2009)
16. Okanohara, D., Sadakane, K.: Practical entropy-compressed rank/select dictionary. In: Proc. 9th ALENEX (2007)
17. Pătraşcu, M.: Lower bounds for 2-dimensional range counting. In: Proc. 39th STOC, pp. 40–46 (2007)
18. Pătraşcu, M., Thorup, M.: Time-space trade-offs for predecessor search. In: Proc. 38th STOC, pp. 232–240 (2006)
19. Raman, R., Raman, V., Srinivasa Rao, S.: Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In: Proc. 13th SODA, pp. 233–242 (2002)
20. Willard, D.: Log-logarithmic worst-case range queries are possible in space $\theta(n)$. Information Processing Letters 17(2), 81–84 (1983)