The Range 1 Query (R1Q) Problem*

Michael A. Bender[1,2], Rezaul Chowdhury[1], Pramod Ganapathi[1], Samuel McCauley[1], and Yuan Tang[3]

[1] Department of Computer Science, Stony Brook University, Stony Brook, NY, USA
{bender,rezaul,pganapathi,smccauley}cs.stonybrook.edu
[2] Tokutek, Inc.
[3] Software School, Fudan University, Shanghai, China
yuantang@csail.mit.edu

Abstract. We define the *range 1 query* (R1Q) problem as follows. Given a d-dimensional ($d \geq 1$) input bit matrix A, preprocess A so that for any given region $\mathcal{R}$ of A, one can efficiently answer queries asking if $\mathcal{R}$ contains a 1 or not. We consider both orthogonal and non-orthogonal shapes for $\mathcal{R}$ including rectangles, axis-parallel right-triangles, certain types of polygons, and spheres. We provide space-efficient deterministic and randomized algorithms with constant query times (in constant dimensions) for solving the problem in the word RAM model. The space usage in bits is sublinear, linear, or near linear in the size of A, depending on the algorithm.

Keywords: R1Q, range query, range emptiness, randomized, rectangular, orthogonal, non-orthogonal, triangular, polygonal, circular, spherical.

1 Introduction

Range searching is one of the fundamental problems in computational geometry [1, 19]. It arises in application areas including geographical information systems, computer graphics, computer aided design, spatial databases, and time series databases. Range searching encompasses different types of problems, such as range counting, range reporting, emptiness queries, and optimization queries.

The *range 1 query* (R1Q) problem is defined as follows. Given a d-dimensional ($d \geq 1$) input bit matrix A (consisting of 0's and 1's), preprocess A so that one can efficiently answer queries asking if any given range $\mathcal{R}$ of A is empty (does not contain a 1) or not, denoted by $\text{R1Q}_A(\mathcal{R})$ or simply $\text{R1Q}(\mathcal{R})$. In 2-D, the range $\mathcal{R}$ can be a rectangle, a right triangle, a polygon or a circle.

In this paper, we investigate solutions in the word RAM model sharing the following characteristics. First of all, we want queries to run in constant time, even for $d \geq 2$ dimensions. Second, we are interested in solutions that have space linear or sublinear in the number of bits in the input grid. Note that while our sublinear bounds are parameterized by the number of 1s in the grid, this is still larger

* Rezaul Chowdhury & Pramod Ganapathi are supported in part by NSF grant CCF-1162196. Michael A. Bender & Samuel McCauley are supported in part by NSF grants IIS-1247726, CCF-1217708, CCF-1114809, and CCF-0937822.

than the information-theoretic lower bounds. For our motivating applications, information-theoretically optimal space is less important than constant query times. Third, we are interested in grid inputs [15, 16], viewing the problem in terms of pixels/voxels rather than a set of spatial points. This grid perspective enables constant-time operations such as table lookup and hashing. Finally, we are interested in both orthogonal and nonorthogonal queries, and we require solutions that are concise enough to be implementable.

Previous Results. The R1Q problem can be solved using data structures such as balanced binary search trees, kd-trees, quad trees, range trees, partition trees, and cutting trees (see [9]), which take the positions of the 1-bits as input. It can also be solved using a data structure of Overmars [16], which uses priority search trees, y-fast tries, and q-fast tries and takes the entire grid as input. However, in d-D ($d \geq 2$), in the worst case these data structures have a query time at least polylogarithmic and occupy a near-linear number of bits.

The R1Q problem can also be solved via range partial sum [7, 21] and the range minimum query (RMQ) [2–6, 10–12, 17, 18, 22] problems. Though several efficient algorithms have been developed to solve the problem in 1-D and 2-D, their generalizations to 3-D and higher dimensions occupying a linear number of bits are not known yet. Also, there is little work on space-efficient constant-time RMQ solutions for non-orthogonal ranges.

The R1Q problem can also be solved using rank queries [13, 14]. Again, its generalization to 2-D and higher dimensions has not yet been studied.

Motivation. We encountered the R1Q and R0Q (whether a range contains a 0) problems while trying to optimize stencil computations in the *Pochoir* stencil compiler [20], where we had to answer octagonal R1Q and octagonal R0Q on a static 2-D property grid. Stencil computations have applications in physics, computational biology, computational finance, mechanical engineering, adaptive statistical design, weather forecasting, clinical medicine, image processing, quantum dynamics, oceanic circulation modeling, electromagnetics, multigrid solvers, and many other areas (see the references in [20]).

In Fig. 1, we provide a simplified exposition of the problem encountered in Pochoir. There are two grids of the same size: a static property grid and a dynamic value grid. Each property grid cell is set to 1 if it satisfies property $\mathcal{P}$ and 0 otherwise. When Pochoir needs to update a range $\mathcal{R}$ in the value grid (see Alg. 1), its runtime system checks whether all or none of the points in $\mathcal{R}$ satisfy $\mathcal{P}$ in the property grid, and based on the query result it uses an appropriate precompiled optimized version of the original code (see Algs. 3, 4) to update the range in the value grid. To check if all points in $\mathcal{R}$ satisfy $\mathcal{P}$, Pochoir uses R0Q($\mathcal{R}$), and to check if no points in $\mathcal{R}$ satisfy $\mathcal{P}$, it uses R1Q($\mathcal{R}$).

Pochoir needs time-, space-, and cache-efficient data structures to answer R1Q. It can also tolerate some false-positive errors. The solutions should achieve constant query time and work in all dimensions. Although it is worth trading off space to achieve constant query times, space is still a scarce resource.

Algorithm 1 : UPDATERANGE($\mathcal{R}$)

```
1: if !R0Q(R) then
2:     {all points in R satisfy P.}
3:     funcptr ← PUPDATEPOINT
4: else if !R1Q(R) then
5:     {no points in R satisfy P.}
6:     funcptr ← NUPDATEPOINT
7: else
8:     {not all points in R satisfy P.}
9:     funcptr ← UPDATEPOINT
10: for each grid point p in R do
11:     funcptr(p)
```

Algorithm 2 : UPDATEPOINT(p)

```
1: {update p only if it satisfies P.}
2: if p.property = 1 then
3:     p.value ← new value
4: do some stuff
```

Algorithm 3 : PUPDATEPOINT(p)

```
1: {p satisfies P. update p.}
2: p.value ← new value
3: do some stuff
```

Algorithm 4 : NUPDATEPOINT(p)

```
1: {p doesn't satisfy P. don't update p.}
2: do some stuff
```

Fig. 1. Examples of the procedures in Pochoir that make use of R1Q and R0Q.

Our Contributions. We solve the R1Q problem for orthogonal and non-orthogonal ranges. Our major contributions as shown in Table 1 are as follows:

1. [Orthogonal Deterministic.] We present a deterministic data structure to answer R1Q for orthogonal ranges in all dimensions and for any data distribution. It occupies linear space in bits and answers queries in constant time for any constant dimension.
2. [Orthogonal Randomized.] We present randomized data structures to answer R1Q for orthogonal ranges. The structures occupy sublinear space in bits and provide a tradeoff between query time and error probability.
3. [Non-Orthogonal Deterministic.] We present deterministic data structures to answer R1Q for non-orthogonal shapes such as axis-parallel right-triangles (for 2-D) and spheres (for all dimensions). The structures occupy near-linear space in bits and answer queries in constant time.

We use techniques such as *power hyperrectangles*, *power right-triangles*, *sketches*, *sampling*, *the four Russians trick*, and *compression* in our data structures. A careful combination of these techniques allows us to solve a large class of R1Q problems. Techniques such as power hyperrectangles, table lookup, and the four Russians trick are already common in RMQ-style operations, while sketches, power right-triangles, and compression are not.

Organization of the Paper. Section 2 presents deterministic and randomized algorithms to answer orthogonal R1Qs on a grid in constant time for constant dimensions. Section 3 presents deterministic algorithms to answer non-orthogonal R1Qs on a grid, for axis-parallel right triangles, some polygons, and spheres.

2 Orthogonal Range 1 Queries (R1Q)

In this section, we present deterministic and randomized algorithms for answering orthogonal R1Qs in constant time and up to linear space.

Shape	Space (in bits)	Time	Comments
Orthogonal (Deterministic)			
d-D	$\mathcal{O}\left((d+1)!\left(\frac{2}{\ln 2}\right)^d N\right) + \lvert A\rvert$	$\mathcal{O}\left(4^d d\right)$	for d dimensions
Orthogonal (Randomized)			
1-D (Sketch)	$\mathcal{O}\left(\sqrt{NN_1}\log N \log\frac{1}{\delta}\right)$	$\mathcal{O}\left(\ln\frac{1}{\delta}\right)$	$\delta \in \left(0, \frac{1}{4}\right)$; correct for range size $\geq \sqrt{N/N_1}$, otherwise correct with prob $\geq 1-4\delta$; extendible to $\geq$ 2-D
1-D (Sketch)	$\mathcal{O}\begin{pmatrix} N_1 \log^2 N \log_{1+\gamma}\frac{1}{\delta} \\ +N^{\frac{1}{c}}\log\log N \end{pmatrix}$	$\mathcal{O}\left(\log\frac{c}{\delta}\right)$	$\gamma, \delta \in \left(0, \frac{1}{4}\right)$, integer $c > 1$; with prob $\geq 1-4\delta$ at most 4γ fraction of all query results will be wrong; extendible to $\geq$ 2-D
1-D (Sampling)	$\mathcal{O}(s) + \lvert A\rvert$	$\mathcal{O}\left(\frac{1}{\varepsilon}\ln\frac{1}{\delta}\right)$	$\varepsilon, \delta \in (0,1)$, $s = \Omega(\log N)$; always correct for range size $\geq (N\log N)/s$, otherwise correct with prob $\geq 1-\delta$ when $\geq \varepsilon$ fraction of all range entries are 1; extendible to $\geq$ 2-D
Non-Orthogonal (Deterministic)			
Right Triangles	$\mathcal{O}\left(N\log N + N_0\log^2 N\right)$	$\mathcal{O}(1)$	not extendible to $\geq$ 3-D
2-D Spheres	$\mathcal{O}\left(N\sqrt{\log N}\right)$	$\mathcal{O}(1)$	extendible to $\geq$ 3-D

Table 1. R1Q algorithms in this paper. Here, N = total #bits, N_1 = #nonzero bits, and N_0 = #zero bits in the input bit matrix A, and d = #dimensions. If $|A|$ appears in the space complexity, it means that A must be retained, otherwise it can be discarded.

The algorithms in this paper rely on finding the most significant bit (MSB) of positive integers in constant time and sublinear space as follows:

Theorem 1. *Given integers $N \in [1, 2^w)$ and $r \in [1, w]$ in the word-RAM model with w-bit words, one can construct a table occupying $\mathcal{O}\left(N^{1/r}\log\log N\right)$ bits of space to answer MSB queries for integers in $[1, N]$ in $\mathcal{O}(1+\log r)$ time.*

2.1 Preliminaries: Deterministic 1-D Algorithm

The input is a bit vector $A[0 \ldots N-1]$, where $N \in [1, 2^w)$ and w is the word size. The query $\text{R1Q}_A(i, j)$, where $i \leq j$, asks if there exists a 1 in the subarray $A[i \ldots j]$. For simplicity, assume N is an even power of 2.

Preprocessing. Array A has $M = \frac{N}{w}$ words. For each $p \in [0, \log M]$, we construct arrays: L_p and R_p, of size $\frac{M}{2^p}$ each. Let $W(i)$ denote the ith ($i \in [0, M-1]$) word in A. Then, L_p is defined as follows: $L_0[i]$ is 0, if $W(i)$ has a 1, 1 otherwise.

$$L_{p(\geq 1)}[i] = \begin{cases} L_{p-1}[2i] & \text{if } L_{p-1}[2i] < 2^{p-1}. \\ 2^{p-1} + L_{p-1}[2i+1] & \text{otherwise.} \end{cases}$$

The R_p array can be computed similarly. The array element $L_p[i]$ (and $R_p[i]$) stores the distance of the leftmost (respectively, rightmost) word that contains a 1 in the ith block of 2^p contiguous words of A, measured from the start (and end) of the block. The value $L_p[i] = 2^p$ ($R_p[i] = 2^p$) means that the ith block of 2^p contiguous words of A does not contain a 1.

Query Execution. To answer $\text{R1Q}_A(i,j)$, we consider two cases: (1) *Intra-word queries:* If (i,j) lies inside one word, we answer R1Q using bit shifts. (2) *Inter-word queries:* If (i,j) spans multiple words, then the query gets split into three subqueries: (a) R1Q from i to the end of its word, (b) R1Q of the words between i's and j's word (both exclusive), and (c) R1Q from the start of j's word to j.

The answer to an inter-word query is 1 if and only if the R1Q for at least one of the three subqueries is 1. The first and third subqueries are intra-word queries and can be answered using bit shifts. Let the words containing indices i and j be I and J, respectively. Then, the second subquery, denoted by $\text{R1Q}_{L_0}(I+1, J-1)$, is answered as follows. Using the MSB of $J-I-1$, we find the largest integer p such that $2^p \leq J-I-1$. The query $\text{R1Q}_{L_0}(I+1, J-1)$ is then decomposed into the following two overlapping queries of size 2^p each: $\text{R1Q}_{L_0}(I+1, I+2^p)$ and $\text{R1Q}_{L_0}(J-2^p, J-1)$. If either of those two ranges contains a 1 then the answer to the original query will be 1, and 0 otherwise. We show below how to answer $\text{R1Q}_{L_0}(I+1, I+2^p)$. Query $\text{R1Q}_{L_0}(J-2^p, J-1)$ is answered similarly.

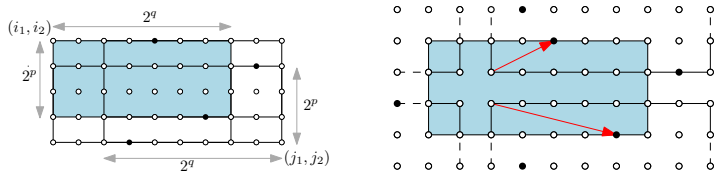
Split L_0 into blocks of size 2^p. Then, the range $\text{R1Q}_{L_0}(I+1, I+2^p)$ can be covered by one or two consecutive blocks. Let $I+1$ be in the kth block. If the range lies in one block, we find whether a 1 exists in that block by checking whether $L_p[k] < 2^p$ is true. If the range is split across two consecutive blocks, we find whether a 1 exists in at least one of the two blocks by checking whether at least one of $R_p[k] \leq (k+1)2^p - I$ or $L_p[k+1] \leq I + 2^p - (k+1)2^p$ is true.

2.2 Deterministic d-D Algorithm

For d-D ($d \geq 2$) R1Q, the input is a bit matrix A of size $N = n^d$. Here we give an algorithm for a 2-D matrix of size $N = n \times n$, but the algorithm extends to higher dimensions. For simplicity, we assume n is a power of 2. The query $\text{R1Q}([i_1, j_1][i_2, j_2])$ asks if there exists a 1 in the submatrix $A[i_1 \ldots j_1][i_2 \ldots j_2]$.

Preprocessing. For each $p, q \in [0, \log n]$, we partition A into $\frac{n}{2^p} \times \frac{n}{2^q}$ blocks, each of size $2^p \times 2^q$ called a (p,q)-block. For each (p,q) pair, we construct four tables of size $\frac{N}{2^{p+q}} \times \min(2^p, 2^q)$ each:
(i) $TL_{p,q}$: if $p \leq q$, $TL_{p,q}[i,j][k]$ indicates that any rectangle of height $k \in [0, 2^p)$ starting from the top-left corner of the current block must have width at least $TL_{p,q}[i,j][k]$ in order to include at least one 1-bit.
(ii) BL, TR, BR: similar to TL but starts from the bottom-left, top-right and bottom-right corners, respectively.
In all cases, a stored value of $\max(2^p, 2^q)$ indicates that the block has no 1.

Query Execution. Given a query $[i_1, j_1][i_2, j_2]$, we find the largest integers p and q such that $2^p \leq j_1 - i_1 + 1$ and $2^q \leq j_2 - i_2 + 1$. The original query range can then be decomposed into four overlapping (p,q)-blocks, which we call *power rectangles*, each with a corner at one of the four corners of the original rectangle, as in Fig. 2(a). If any of these four rectangles contains a 1, the answer to the original query will be 1, and 0 otherwise. We show below how to answer an R1Q for a power rectangle.



(a) (b)

Fig. 2. Rectangles: (a) Query rectangle split into four possibly overlapping power rectangles. (b) Power rectangle divided into four regions by four split rectangles.

We consider the partition of A into preprocessed (p, q)-blocks. It is easy to see that each of the four power rectangles of size $2^p \times 2^q$ will intersect at most four preprocessed (p, q)-blocks. We call each rectangle contained in both the power rectangle and a (p, q)-block a *split rectangle* (see Fig. 2(b)). The R1Q for a split rectangle can be answered using a table lookup, checking if the table values of the appropriate (p, q)-blocks are inside the power rectangle boundary, as shown in Fig. 2(b). The proof of the following theorem will be given in the full paper.

Theorem 2. *Given a d-D input grid of size $N = n^d$, each orthogonal R1Q on the grid can be answered deterministically in $\mathcal{O}\left(4^d d\right)$ time after preprocessing the grid in $\Theta(N)$ time using $\mathcal{O}\left((d+1)!\,(2/\ln 2)^d\, N\right)$ bits of space. In 1-D, the space can be reduced to $\mathcal{O}(N/\log N)$ bits.*

2.3 Randomized Algorithms

In this section, we present randomized algorithms that build on the deterministic algorithms given in Sections 2.1 and 2.2. We describe the algorithms for one dimension only. Extensions to higher dimensions are straightforward.

Sketch Based Algorithms. Our algorithms provide probabilistic guarantees based on the Count-Min (CM) sketch data structure proposed in [8]. Let N_1 be the number of 1-bits in the input bit array $A[0 \ldots N-1]$ for any data distribution. Then, the (preprocessing) time and space complexities depend on N_1 while the query time remains constant.

A CM sketch with parameters $\varepsilon \in (0, 1]$ and $\delta \in (0, 1)$ can store a summary of any given vector $\boldsymbol{a} = \langle a_0, a_1, \ldots, a_{n-1} \rangle$ with $a_i \geq 0$ in only $\lceil \frac{e}{\varepsilon} \rceil \lceil \ln \frac{1}{\delta} \rceil \log ||\boldsymbol{a}||_1$ bits of space, where $||\boldsymbol{a}||_1$ (or $||\boldsymbol{a}||$) $= \sum_{i=0}^{n-1} a_i$, and can provide an estimate $\hat{a}_i$ of any a_i with the following guarantees: $a_i \leq \hat{a}_i$, and with probability at least $1-\delta$, $\hat{a}_i \leq a_i + \varepsilon ||\boldsymbol{a}||_1$. It uses $t = \lceil \ln \frac{1}{\delta} \rceil$ hash functions $h_1 \ldots h_t : \{0 \ldots n-1\} \rightarrow \{1 \ldots b\}$ chosen uniformly at random from a pairwise-independent family, where bucket size $b = \lceil \frac{e}{\varepsilon} \rceil$. These hash functions are used to update a 2-D matrix $c[1:t][1:b]$ of bt counters initialized to 0. For each $i \in [0, n-1]$ and each $j \in [1, t]$ one then updates $c[j][h_j(i)]$ to $c[j][h_j(i)] + a_i$. After the updates, an estimate $\hat{a}_i$ for any given query point a_i is obtained as $\min_{1 \leq j \leq t} c[j][h_j(i)]$.

Preprocessing. In the deterministic algorithms we first compressed the input array by converting each word into a single bit, and then constructed L_0 and R_0 arrays from the compressed array. In the current algorithm we build the L_0 and R_0 arrays directly from the uncompressed input. For $p \in \left[0, \frac{1}{2}\log\left(N/N_1\right)\right]$, the L_p and R_p arrays are stored as CM sketches while $p \in \left[\frac{1}{2}\log\left(N/N_1\right)+1, \log N\right]$ the arrays are stored directly as in the deterministic case. Each $L_p[i]$ is added as $(L_p[i]+1) \mod (2^p+1)$ to the CM sketch (similarly for $R_p[i]$). Thus a nonzero entry (of value at most 2^p) is added to the CM sketch provided the corresponding block contains a 1, otherwise nothing is added. As a result for any given L_p summation of all entries added to the CM sketch is at most $N_1 \times 2^p$, and we set $\varepsilon = \frac{1}{2\times N_1 \times 2^p}$ for that sketch.

Query Execution. Given a query $\mathrm{R1Q}_A(i,j)$, we use the MSB of $j-i+1$ to find the largest value of p with $2^p \leq j-i+1$, and then follow the approach for answering case (b) of inter-word queries described in Section 2.1. If $2^p > \sqrt{N/N_1}$, we use L_p and R_p arrays to answer the query correctly, otherwise we use the L_p and R_p values obtained from the corresponding CM sketches.

Error Bound. If the query range is larger than $\sqrt{N/N_1}$, the answer is always correct. For smaller queries we use CM sketches. Recall that for $p \in \left[0, \frac{1}{2}\ln(N/N_1)\right]$, we store each L_p (and R_p) as a CM sketch with parameter $\varepsilon = \frac{1}{2\times N_1 \times 2^p}$. Hence, the estimated value $\hat{L}_p[i]$ of an entry $L_p[i]$ returned by the CM sketch is between $L_p[i]$ and $L_p[i] + \varepsilon||L_p|| \leq L_p[i] + 0.5$ with probability at least $1-\delta$. In other words, with probability at least $1-\delta$, the CM sketch returns the correct value. In order to answer an R1Q we need to access at most four CM sketches. Hence, with probability at least $(1-\delta)^4 \geq 1-4\delta$, the query will return the correct answer.

Theorem 3. *Given a 1-D bit array of length N containing N_1 nonzero entries, and a parameter $\delta \in \left(0, \frac{1}{4}\right)$, one can construct a data structure occupying $\mathcal{O}\left(\sqrt{NN_1}\log N \log\left(\frac{1}{\delta}\right)\right)$ bits (and discard the input array) to answer each R1Q correctly in $\mathcal{O}\left(\ln\frac{1}{\delta}\right)$ worst-case time with probability at least $1-4\delta$. For query ranges larger than $\sqrt{N/N_1}$ the query result is always correct.*

By tweaking the algorithm described above slightly, we can reduce the space complexity even further at the cost of providing a weaker correctness guarantee. We assume that we are given an additional parameter $\gamma \in \left(0, \frac{1}{4}\right)$. The required modification is described below.

For each $p \in [0, \log N]$, we store the L_p and R_p arrays as CM sketches. However, instead of adding a value v directly to a CM sketch, we now add a $(1+\gamma)$ approximation of v. More precisely, we add $\lceil \log_{1+\gamma}(1+v) \rceil$ instead of v. Hence, for a given L_p, the summation of all entries added to its CM sketch is at most $N_1 \lceil \log_{1+\gamma}(1+2^p) \rceil$, and so we set the parameter ε to $1/\left(2N_1 \lceil \log_{1+\gamma}(1+2^p) \rceil\right)$ for that sketch. The total space used by all CM sketches can be shown to be $\mathcal{O}\left(N_1 \log^3 N \log_{1+\gamma}(1/\delta)\right)$. We store a lookup table of size $\mathcal{O}\left(\log^2 N\right)$ for conversions from $\lceil \log_{1+\gamma}(1+v) \rceil$ to v, and an MSB table of size $\mathcal{O}\left(N^{1/c}\log\log N\right)$ for some given integer constant $c > 1$.

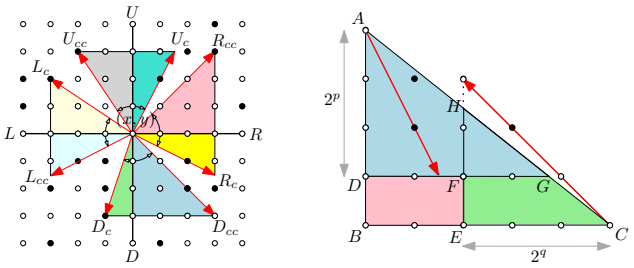
We first show that for any given $p \in [0, \log N]$ at most 2γ fraction of the queries of size 2^p can return incorrect answers. Consider any two consecutive blocks of size 2^p, say, blocks $i \in [0, \frac{N}{2^p} - 1)$ and $i+1$. Exactly 2^p different queries of size 2^p will cross the boundary between these two blocks. The answer to each of these queries will depend on the estimates of $R_p[i]$ and $L_p[i+1]$ obtained from the CM sketches. Under our construction the estimates are $\hat{R}_p[i] \leq (1+\gamma)R_p[i] \leq R_p[i] + \gamma \cdot 2^p$ and $\hat{L}_p[i+1] \leq (1+\gamma)L_p[i+1] \leq L_p[i+1] + \gamma \cdot 2^p$. Hence, at most $\gamma \cdot 2^p$ of those 2^p queries will produce incorrect results due to the error in estimating $R_p[i]$, and at most $\gamma \cdot 2^p$ more because of the error in estimating $L_p[i+1]$. Thus with probability at least $(1-\delta)^2$, at most 2γ fraction of those 2^p queries will return wrong results. Recall from Section 2.1 that we answer given queries by decomposing the query range into two overlapping query ranges. Hence, with probability at least $(1-\delta)^4 \geq 1 - 4\delta$, at most $2\gamma + 2\gamma = 4\gamma$ fraction of all queries can produce wrong answers.

Theorem 4. *Given a 1-D bit array of length N containing N_1 nonzero entries, and two parameters $\gamma \in \left(0, \frac{1}{4}\right)$ and $\delta \in \left(0, \frac{1}{4}\right)$, and an integer constant $c > 1$, one can construct a data structure occupying $\mathcal{O}\left(N_1 \log^3 N \log_{1+\gamma}\left(\frac{1}{\delta}\right) + N^{1/c} \log\log N\right)$ bits (and discard the input array) to answer each R1Q in $\mathcal{O}\left(\log \frac{c}{\delta}\right)$ worst-case time such that with probability at least $1 - 4\delta$ at most 4γ fraction of all query results will be wrong.*

Sampling Based Algorithm. Suppose we are allowed to use only $\mathcal{O}(s)$ bits of space (in addition to the input array A), and $s = \Omega(\log_2 N)$. We are also given two constants $\varepsilon \in (0,1)$ and $\delta \in (0,1)$. We build L_p and R_p arrays for each $p \in \left[\log \frac{N}{s} + \log\log N, \log N\right]$, and an MSB lookup table to support constant time MSB queries for integers in $[1, s/\log N]$. Consider the query $\mathrm{R1Q}_A(i,j)$. If $j - i + 1 \leq w$, we answer the query correctly in constant time by reading at most 2 words from A and using bit shifts. If $j - i + 1 \geq 2^{\log \frac{N}{s} + \log\log N} = \frac{N \log N}{s}$, we use the L_p and R_p arrays to correctly answer the query in constant time. If $w < j - i + 1 < \frac{N \log N}{s}$, we sample $\lceil \frac{1}{\varepsilon} \ln\left(\frac{1}{\delta}\right) \rceil$ entries uniformly at random from $A[i \ldots j]$, and return their bitwise OR. It is easy to show that the L_p and R_p tables use $\mathcal{O}(s)$ bits in total, and the MSB table uses $o(s)$ bits of space. The query time is clearly $\mathcal{O}\left(\frac{1}{\varepsilon} \ln\left(\frac{1}{\delta}\right)\right)$.

Error Bound. If at least an ε fraction of the entries in $A[i \ldots j]$ are nonzero then the probability that a sample of size $\lceil \frac{1}{\varepsilon} \ln\left(\frac{1}{\delta}\right) \rceil$ chosen uniformly at random from the range will pick at least one nonzero entry is $\geq 1 - (1-\varepsilon)^{\frac{1}{\varepsilon} \ln\left(\frac{1}{\delta}\right)} \approx 1 - \delta$.

Theorem 5. *Given a 1-D bit array of length N, a space bound $s = \Omega(\log N)$, and two parameters $\varepsilon \in (0,1)$ and $\delta \in (0,1)$, one can construct a data structure occupying only $\mathcal{O}(s)$ bits of space (in addition to the input array) that in $\mathcal{O}\left(\frac{1}{\varepsilon} \ln\left(\frac{1}{\delta}\right)\right)$ time can answer each R1Q$_A(i,j)$ correctly with probability at least $1 - \delta$ provided at least an ε fraction of the entries in $A[i \ldots j]$ are nonzero. If $j - i + 1 \leq w$ or $j - i + 1 \geq \frac{N \log N}{s}$, the query result is always correct.*



(a) (b)

Fig. 3. Right Triangular R1Q. (*a*) Preprocessing. (*b*) Query Execution. Each black grid point contains a 1 while each white grid point contains a 0.

3 Non-Orthogonal Range 1 Queries (R1Q)

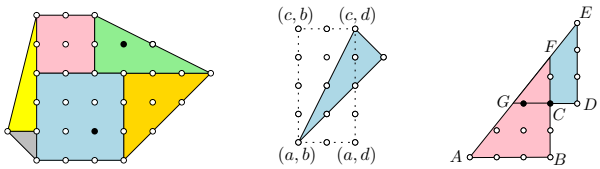
In this section, we show how to answer R1Q for non-orthogonal ranges, such as axis-parallel right triangles, spheres and certain type of polygons, given an input matrix of size $N = n \times n$.

3.1 Right Triangular R1Q

A right triangular query R1Q(ABC) asks if there exists a 1 in an axis-parallel right triangle ABC defined by three grid points A, B, and C. In the rest of the paper, right triangles will mean axis-aligned right triangles.

Preprocessing. For every grid point (x, y) containing a 0, for each $p \in \left[0, \frac{\log N}{2}\right]$, we store the coordinates of 8 other grid points for 8 different orientations. For example, consider Fig. 3(a) in which each black grid point corresponds to a 1, and each white point corresponds to a 0. For the point $P = (x, y)$ in the figure, we show the eight black points (i.e., L_C, L_{CC}, R_C, R_{CC}, U_C, U_{CC}, D_C and D_{CC}) we store for a given p. For example, L_C is a black point that lies to the left of P within a horizontal distance of 2^p from it (in terms of the number of grid points including P) such that PL_C makes the smallest angle θ_{LC} in the clockwise direction with the horizontal line passing through P. The significance of L_C is that no right triangle with a horizontal base of length 2^p that has one endpoint at (x, y), another endpoint to the left of (x, y), and whose hypotenuse makes a smaller nonnegative angle than θ_{LC} in the clockwise direction with the horizontal line can contain a 1. Similarly, other points are identified.

Query Execution. We show how to answer a right triangular R1Q in $\Theta(1)$ time. Say, we want to answer R1Q(ABC) (see Fig. 3(b)). Let 2^p be the largest power of 2 not larger than $|AB|$, and 2^q be the largest power of 2 not larger than $|CB|$. Find grid points D and E on AB and CB, respectively, such that $|AD| = 2^p$ and $|CE| = 2^q$. Suppose the horizontal line passing through D intersects BC at G, and the vertical line passing through E intersects BC at H. Observe that G and H are not necessarily grid points. We assume w.l.o.g. that none of the vertices A, B and C contains a 1 (as otherwise we can answer the query trivially



(a) (b) (c)

Fig. 4. Black grid points contain 1's and white grid points contain 0's. Polygon in (a) satisfies Prop. 1. Polygons in (b) and (c) do not satisfy Prop. 1. Still, R1Q can be answered for (c).

in constant time). Observe that we can answer R1Q(ABC) if we can answer R1Q for triangles ADG and CEH, and the rectangle $BDFE$. R1Q for the rectangle can be answered using our deterministic algorithm described in Section 2.2. R1Q for a right triangle of a particular orientation with height or base length equal to a power of two can be answered in constant time. This is done by checking whether the point stored (from preprocessing) with the appropriate endpoint of the hypotenuse for that specific orientation is inside the triangle or not.

Theorem 6. *Given a 2-D bit matrix of size $N = \sqrt{N} \times \sqrt{N}$ containing N_0 zero bits, one can construct a data structure occupying $\mathcal{O}\left(N \log N + N_0 \log^2 N\right)$ bits in $\mathcal{O}\left(N^{1.5}\right)$ time (and discard the input matrix) to answer each axis-aligned right triangular R1Q with the three vertices on the grid points in $\mathcal{O}(1)$ time.*

3.2 Polygonal R1Q

Consider a simple polygon with its vertices on grid points satisfying the following.

Property 1. For every two adjacent vertices (a, b) and (c, d), one of the two right triangles with the third vertex being either (a, d) or (c, b) is completely inside the polygon.

It can be shown that such a polygon can be decomposed into a set of possibly overlapping right triangles and rectangles with only grid points as vertices that completely covers the polygon (see Fig. 4(a)). Examples of polygons that do not satisfy the constraint are given in Fig. 4(b, c), but we can still answer R1Q for the polygon in (c). A simple polygon with k vertices satisfying propery 1 can be decomposed into $\mathcal{O}(k)$ right triangles and rectangles and hence can be answered in $\mathcal{O}(k)$ time.

3.3 Spherical R1Q

The spherical R1Q problem is defined as follows. Given a d-dimensional ($d \geq 2$) input bit matrix A, preprocess A such that given any grid point p in A and a radius $r \in \mathbb{R}^+$, find efficiently if there exists a 1 in the d-sphere centered at p of radius r. Here, we present the algorithm for 2-D. The approach can be extended to higher dimensions.

A nearest 1-bit of a grid point p is called a *nearest neighbor* (NN) of p. We preprocess A by computing and compresssing the NNs of all grid points in A

(only one NN per grid point). We then answer a spherical R1Q by checking whether a NN of the given center point is inside the circle of given radius.

Preprocessing. We store the locations of the NNs of the grid points of A in a temporary NN matrix that occupies $\mathcal{O}(N \log N)$ bits, but can be compressed to occupy $\mathcal{O}\left(N\sqrt{\log N}\right)$ bits as follows.

We divide the grid into $\sqrt{(\log N)/6} \times \sqrt{(\log N)/6}$ blocks. We store the NN position for all points on the boundary of each block. The interior points will be replaced with arrows $(\rightarrow, \leftarrow, \uparrow, \downarrow)$ and bullets $(\bullet)$ as follows. If a grid point p contains a 1 then p is replaced with a $\bullet$. An arrow at a grid point gives the direction of its NN. If we follow the arrows from any interior point, we end up in either a boundary point or an interior point containing a 1. For any given block the matrix created as above will be called a *symbol matrix* representing the block.

Two blocks are of the same *type* if they have the same symbol matrix. Each symbol can be represented using only three bits. Since each block has $(\log N)/6$ symbols, there are $2^{\frac{3 \log N}{6}} = \sqrt{N}$ possible block types. For each block type we create a *position matrix* that stores, for each grid point within a block, the pointer to its NN if the NN is an interior point, or a pointer to a boundary point if the NN is an exterior or boundary point. The boundary point will have stored its own NN position in the input array.

We can now discard the original input matrix, and replace it with the following compressed representation. For each block in the input matrix we store its block type (i.e., a pointer to the corresponding block type) followed by the NN positions of its boundary points. For each block type we retain its position matrix.

Query Execution. We can answer a spherical R1Q by checking whether the NN position of the center point is inside the query sphere. The approach of finding the NN position is as follows. We find the block to which the given point belongs and follow the pointer to its block type. We check the position stored at the given point in the position matrix. If it points to an internal point, then that point is the correct NN. If it points to a boundary point, we again follow the pointer stored at the boundary point to get the correct NN.

Theorem 7. *Given a 2-D bit array of size N, one can construct a data structure occupying $\mathcal{O}\left(N\sqrt{\log N}\right)$ bits (and discard the input array) to answer each spherical R1Q in $\mathcal{O}(1)$ time.*

Acknowledgments. We like to thank Michael Biro, Dhruv Matani, Joseph S. B. Mitchell, and anonymous referees for insightful comments and suggestions.

References

1. Agarwal, P.K., Erickson, J.: Geometric range searching and its relatives. Contemporary Mathematics 223, 1–56 (1999)
2. Amir, A., Fischer, J., Lewenstein, M.: Two-dimensional range minimum queries. In: CPM. pp. 286–294. Springer (2007)

3. Bender, M.A., Farach-Colton, M.: The lca problem revisited. In: LATIN, pp. 88–94. Springer (2000)
4. Bender, M.A., Farach-Colton, M., Pemmasani, G., Skiena, S., Sumazin, P.: Lowest common ancestors in trees and directed acyclic graphs. Journal of Algorithms 57(2), 75–94 (2005)
5. Berkman, O., Vishkin, U.: Recursive star-tree parallel data structure. SIAM Journal on Computing 22(2), 221–242 (1993)
6. Brodal, G.S., Davoodi, P., Rao, S.S.: On space efficient two dimensional range minimum data structures. Algorithmica 63(4), 815–830 (2012)
7. Chazelle, B., Rosenberg, B.: Computing partial sums in multidimensional arrays. In: SoCG. pp. 131–139. ACM (1989)
8. Cormode, G., Muthukrishnan, S.: An improved data stream summary: the count-min sketch and its applications. Journal of Algorithms 55(1), 58–75 (2005)
9. De Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational geometry. Springer (2008)
10. Fischer, J.: Optimal succinctness for range minimum queries. In: LATIN, pp. 158–169. Springer (2010)
11. Fischer, J., Heun, V.: A new succinct representation of rmq-information and improvements in the enhanced suffix array. In: Combinatorics, Algorithms, Probabilistic and Experimental Methodologies, pp. 459–470. Springer (2007)
12. Fischer, J., Heun, V., Stiihler, H.: Practical entropy-bounded schemes for o (1)-range minimum queries. In: Data Compression Conference. pp. 272–281. IEEE (2008)
13. Golynski, A.: Optimal lower bounds for rank and select indexes. TCS 387(3), 348–359 (2007)
14. González, R., Grabowski, S., Mäkinen, V., Navarro, G.: Practical implementation of rank and select queries. In: Poster Proc. WEA. pp. 27–38 (2005)
15. Navarro, G., Nekrich, Y., Russo, L.: Space-efficient data-analysis queries on grids. TCS (2012)
16. Overmars, M.H.: Efficient data structures for range searching on a grid. Journal of Algorithms 9(2), 254–275 (1988)
17. Sadakane, K.: Compressed suffix trees with full functionality. Theory of Computing Systems 41(4), 589–607 (2007)
18. Sadakane, K.: Succinct data structures for flexible text retrieval systems. JDA 5(1), 12–22 (2007)
19. Sharir, M., Shaul, H.: Semialgebraic range reporting and emptiness searching with applications. SIAM Journal on Computing 40(4), 1045–1074 (2011)
20. Tang, Y., Chowdhury, R., Kuszmaul, B.C., Luk, C.K., Leiserson, C.E.: The Pochoir stencil compiler. In: SPAA. pp. 117–128. ACM (2011)
21. Yao, A.C.: Space-time tradeoff for answering range queries. In: STOC. pp. 128–136. ACM (1982)
22. Yuan, H., Atallah, M.J.: Data structures for range minimum queries in multidimensional arrays. In: SODA. pp. 150–160 (2010)