Succinct Indices for Range Queries with applications to Orthogonal Range Maxima*

Arash Farzan[1], J. Ian Munro[2], and Rajeev Raman[3]

[1] Max-Planck-Institut für Informatik, Saarbücken, Germany.
[2] University of Waterloo, Canada.
[3] University of Leicester, UK.

Abstract. We consider the problem of preprocessing N points in 2D, each endowed with a priority, to answer the following queries: given a axis-parallel rectangle, determine the point with the largest priority in the rectangle. Using the ideas of the *effective entropy* of range maxima queries and *succinct indices* for range maxima queries, we obtain a structure that uses $O(N)$ words and answers the above query in $O(\lg N \lg\lg N)$ time. This a direct improvement of Chazelle's result from 1985 [10] for this problem – Chazelle required $O(N/\epsilon)$ words to answer queries in $O((\lg N)^{1+\epsilon})$ time for any constant $\epsilon > 0$.

1 Introduction

Range searching is one of the most fundamental problems in computer science with important applications in areas such as computational geometry, databases and string processing. The input is a set of N points in general position in $\mathbb{R}^d$ (we focus on the case $d = 2$), where each point is associated with *satellite* data, and an aggregation function defined on the satellite data. We wish to preprocess the input to answer queries of the following form efficiently: given any 2D axis-aligned rectangle R, return the value of the aggregation function on the satellite data of all points in R. Researchers have considered range searching with respect to diverse aggregation functions such as emptiness checking, counting, reporting, minimum/maximum, etc. [10, 12, 17]. In this paper, we consider the problem of *range maximum* searching (the minimum variant is symmetric), where the satellite data associated with each point is a numerical *priority*, and the aggregation function is "arg max", i.e., we want to report the point with the maximum priority in the given query rectangle. This aggregation function is *the* canonical one to study, among those that do not admit inverses [10].

Our primary concern is to obtain *linear-space* data structures, namely those that occupy $O(N)$ words, and we seek to minimize query time subject to this constraint. The space usage is a fundamental concern in geometric data structures due to very large data volumes; indeed, space usage is a main reason why range searching data structures like quadtrees, which have poor worst-case query performance, are preferred in many practical applications over data structures such

* Work done while Farzan was employed by, and Raman was visiting, MPI.

Citation	Size (in words)	Query time
Chazelle'88 [10]	$O(N \lg^{\epsilon} N)$	$O(\lg N)$
Chan et al.'10 [8]	$O(N \lg^{\epsilon} N)$	$O(\lg \lg N)$
Karpinski et al.'09 [15]	$O\Big(N(\lg \lg N)^{O(1)}\Big)$	$O\big((\lg \lg N)^2\big)$
Chazelle'88 [10]	$O(N \lg \lg N)$	$O(\lg N \lg \lg N)$
Chazelle'88 [10]	$O\big(\frac{1}{\epsilon}N\big)$	$O\big(\lg^{1+\epsilon} N\big)$
NEW	$O(N)$	$O(\lg N \lg \lg N)$

Table 1. Space/time tradeoffs for 2D range maximum searching in the word RAM.

as range trees, which have asymptotically optimal query performance. Space efficient solutions to range searching date to the work of Chazelle [10] over a quarter century ago, and Nekrich [18] gives a nice survey of much of this work. Recently there has been a flurry of activity on various aspects of space-efficient range reporting, and for some aggregation functions there has even been attention given to the constant term within the space usage [5, 18].

We now formalize the problem studied by our paper, as well as those of [10, 8, 15]. We assume input points are in *rank space*: the x-coordinates of the n points are $\{0, \ldots, N-1\} = [N]$, and the y-coordinates are given by a permutation $\upsilon : [N] \to [N]$, such that the points are $(i, \upsilon(i))$ for $i = 0, \ldots, N-1$. The priorities of the points are given by another permutation π such that $\pi(i)$ is the priority of the point $(i, \upsilon(i))$. The reduction to rank space can be performed in $O(\lg N)$ time with a linear space structure even if the original and query points are points in $\mathbb{R}^2$ [12, 10]. The query rectangle is specifed by two points from $[N] \times [N]$ and includes the boundaries (see Fig. 1(R)). Analogous to previous work, we also assume the word-RAM model with word size $\Theta(\lg N)$ bits[4].

Range maximum searching is a well-studied problem (Table 1). Chazelle [10] gave a few space/time tradeoffs covering a broad spectrum. To the best of our knowledge, the solution with the lowest query time that uses only $O(N)$ words is that of Chazelle [10], who gave a data structure of size $O(\frac{1}{\epsilon}N)$ words with query time $O(\lg^{1+\epsilon} N)$ for any fixed $\epsilon > 0$. More recent results on the range maximum problem are as follows. Karpinski *et al.* [15] studied the problem of 3D five-sided range emptiness queries which is closely related to range maximum searching in 2D. As observed in [8], their solution yields a query time of $(\lg \lg N)^{O(1)}$ with an index of size $N(\lg \lg N)^{O(1)}$ words. Chan *et al.* [8] currently give the best query time of $O(\lg \lg N)$, but this is at the expense of using $O(N \lg^{\epsilon} N)$ words, for any fixed $\epsilon > 0$. However, there has been no improvement in the running time for linear-space data structures. In this paper, we improve Chazelle's long-standing result by giving a data structure of $O(N)$ words and reducing the query time from polylogarithmic to "almost" logarithmic, namely, $O(\lg N \lg \lg N)$.

Although our primary focus is on 4-sided queries, which specify a rectangle that is bounded from all sides, we also need to consider 2-sided and 3-sided

[4] $\lg x = \log_2 x$

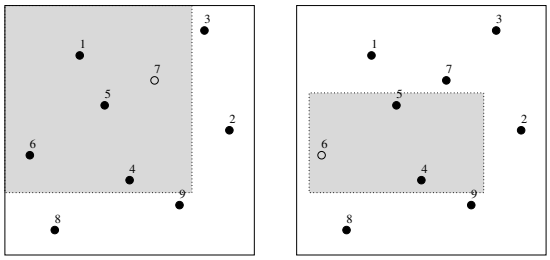

Fig. 1. 2-sided and 4-sided range maximum queries. The numbers with the points represent their priorities, and the unshaded points are the answers.

queries, which are "open" on two and one side respectively (thus a 2-sided query is specified by a single point (i, j)—see Fig. 1(L)—and a 3-sided query by two points (i, j) and (k, l) where either $i = k$ or $j = l$). Our solution recursively divides the points into horizontal and vertical slabs, and a query rectangle is decomposed into smaller 2-sided, 3-sided, and 4-sided queries. A key intermediate result is the data structure for 2-sided queries. The 2-sided sub-problems are partitioned into smaller sub-problems, which are stored in a "compressed" format that is "decompressed" at query time. The "compression" uses the idea that to answer 2-sided range maxima queries on a problem of size m, one need not store the entire total order of priorities using $\Theta(m \lg m)$ bits: $O(m)$ bits suffice, i.e., the *effective entropy* [13] of 2-sided queries is low. This does not help immediately, since $\Theta(m \lg m)$ bits are needed to store the coordinates of the points comprising these sub-problems. To overcome this bottleneck, the data structures for these sub-problems are *succinct indices* [2]: they do not store the coordinates, and instead obtain them from a global data structure during "decompression". We solve 3-sided and 4-sided subqueries by recursion, or by using succinct indices for range maximum queries on matrices [20, 6]. When recursing, we cannot afford the space required to store the structures for rank space reduction for each such subproblem: a further key idea is to use a single global structure to achieve this.

By reusing ideas from this 2-sided result, we obtain two stand-alone results on succinct indices for 2-sided range maxima queries. We show that given N points in rank space, together with priorities, it is possible to answer 2-sided range maxima queries in just $O(N)$ additional bits (i.e excluding point coordinate information) in $(\lg N)^{O(1)}$ time, assuming the index can access point coordinates via an orthogonal range reporting [12, 8] queries. This result has been recently used in a context where the input is a low-discrepancy point set whose coordinates need not be stored at all, and can be generated "on the fly". A different index for the *permuted-point* model of Bose et al. [4] uses $O(N)$ additional bits and answers 2-sided range maxima queries in only $O(\lg \lg N)$ time.

The paper is organized as follows. We first describe some building blocks used in Section 2. Section 3 is devoted to our main result, and Section 4 describes the succinct index results. Many proofs have been omitted from this extended abstract and may be found in [11].

2 Preliminaries

In order to support mapping between recursive sub-problems, we use the following primitives on a set S of N points in rank space. A *range counting* query reports the *number* of points within a query rectangle:

Lemma 1 ([14]). *Given a set of N points in rank space in two dimensions, there is a data structure with $O(N)$ words of space that supports range counting queries in $O(\lg N / \lg \lg N)$ time.*

A *range reporting* structure supports the operation of listing the coordinates of all points within a query rectangle. We use the following consequence of a result of Chan *et al.* [8]:

Lemma 2 ([8]). *Given a set of N points in rank space in two dimensions, there is a data structure with $O(N)$ words of space that supports range reporting queries in $O\Big((1+k)\lg^{1/3} N\Big)$ time where k is the number of points reported.*

The *range selection* problem is as follows: given an input array A of size N, to preprocess it so that given a query (i, j, k), with $1 \leq i \leq j \leq N$, we return an index i_1 such that $A[i_1]$ is the k-th smallest of the elements in the subarray $A[i], A[i+1], \ldots, A[j]$.

Lemma 3 ([7]). *Given an array of size N, there is a data structure with $O(N)$ words of space that supports range selection queries in $O(\lg N / \lg \lg N)$ time.*

3 The data structure

In this section we show our main result:

Theorem 1. *Given N points in two-dimensional rank space, and their priorities, there is a data structure that occupies $O(N)$ words of space and answers range maximum queries in $O(\lg N \lg \lg N)$ time.*

We first give an overview of the data structure. We begin by storing all the points (using their input coordinates) once each in the structures of Lemmas 1 and 2. We also store an instance of the data structure of Lemma 3 once each for the arrays X and Y, where $X[i] = \nu(i)$ and $Y[i] = \nu^{-1}(i)$ for $i \in N$ (X stores the y-coordinates of the points in order of increasing x-coordinate, and Y the x-coordinates in order of increasing y-coordinate). These four "global" data structures use $O(N)$ words of space in all.

We recursively decompose the problem à la Afshani *et al.* [1]. Let n be the recursive problem size (initially $n = N$). Given a problem of size n, we divide the problem into n/k mutually disjoint horizontal *slabs* of size n by k, and n/k mutually disjoint vertical slabs of size k by n. A horizontal and vertical slab intersect in a *square* of size $k \times k$. We recurse on each horizontal or vertical slab: observe that each horizontal or vertical slab has exactly k points in it, and is treated as a problem of size k—i.e. it is logically comprised of two permutations

υ and π on $[k]$ (Fig. 2(L); Sec. 3.1). Given a slab in a problem of size n containing k points, we need to map coordinates in the slab (which in one dimension will be from $[n]$) down to $[k] \times [k]$ in order to view the slab as a problem of size k—and back again. This mapping is *not* explicitly stored, and is achieved through *slab-rank* and *slab-select* operations (Sec. 3.2).

The given query rectangle is decomposed into a number of disjoint recursive 2-sided, 3-sided and 4-sided queries. In addition to queries that reach slabs at the bottom of the recursion, many other queries do not generate further recursive problems. Such queries are called *terminal*, and the problems (or data structures) that involve answering terminal queries are also called terminal. Each terminal query produces some *candidate* points: the set of all candidate points must contain the final answer. To achieve the space bound, we require that all terminal problems of size n—except those at the bottom of the recursion—use space $O(n\sqrt{\lg n})$ bits (Sec. 3.1). Terminal 3- and 4-sided problems are handled by the results of [20, 6]. Terminal 2-sided problems reduce the range maximum query to planar point location, but the space bound precludes an explicit representation. Instead, the data structures are *succinct indices*—the points that comprise them are accessed by means of queries to a single global data structure. Using the key insight that $O(n)$ bits suffice to encode the priority information needed to answer 2-sided queries in a problem of size n (Sec. 3.3), we store parts of the planar sub-division in the recursive problems in a compressed form, relevant parts of which are recomputed at query time (Sec. 3.4).

3.1 A recursive formulation and its space usage

The recursive structure is as follows. Let $L = \lg N$, and consider a recursive problem of size n (at the top level $n = N$). We assume wlog that N and n are powers of 2, as are a number of expressions which represent the size of recursive problems (if not, replace real-valued parameters $x \geq 1$ by $2^{\lfloor \lg x \rfloor}$ or $2^{\lceil \lg x \rceil}$ without affecting the asymptotic complexity). Unless we have reached the bottom of the recursion, we partition the input range $[n] \times [n]$ into mutually disjoint vertical slabs of width $k = \sqrt{nL}$ and also into mutually disjoint horizontal slabs of height $k = \sqrt{nL}$, which intersect in $O(n/L)$ $k \times k$ squares. We need to answer 2-sided, 3-sided or 4-sided queries on this problem, which we do as follows (see Fig. 2(R)):

- A 2-sided query is terminal and generates one candidate.
- A 3-sided query results in at most one recursive 3-sided query on a slab, plus up to three terminal problems, each generating one candidate: at most two 2-sided queries on slabs, and at most one *square-aligned* 3-sided query (a square-aligned query exactly covers a rectangular sub-array of squares).
- A 4-sided query either results in a recursive 4-sided query on a slab or results at most one square-aligned 4-sided query (generating one candidate), plus up to four recursive 3-sided queries in slabs.

Since each 3-sided query only generates one recursive 3-sided query, $O(r) = O(\lg \lg N)$ recursive problems are solved generating $O(r)$ candidates.

The data structures associated with the current recursive problem are:

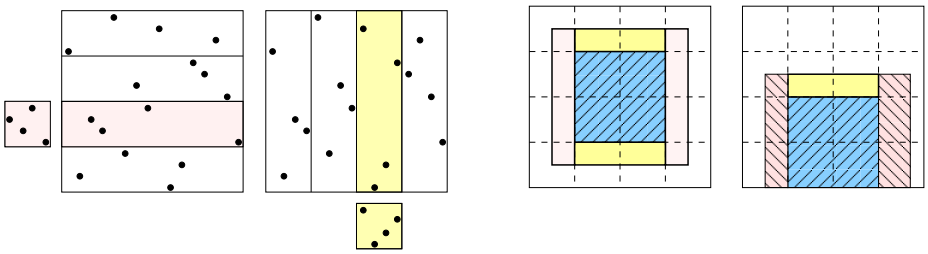

Fig. 2. The recursive decomposition of the input (L) and queries (R). In (R) shaded problems are terminal problems.

- for problems at the bottom of the recursion, we store an instance of Chazelle's data structure which uses $O(n \lg n)$ bits of space and has query time $O\big((\lg n)^2\big)$.
- For 2-sided queries (which are terminal) in non-terminal problems we use the data structure with space usage $O(n\sqrt{\lg n})$ bits described in Sec. 3.4.
- For 3- and 4-sided square-aligned queries, we store an $n/k \times n/k$ matrix that contains the (top-level) coordinates and priority of the maximum point (if any) in each square. This matrix uses $O(n/L \cdot L) = O(n)$ bits. We then use the data structure of [20, 6] for answering 2D range maximum queries on the elements in the above matrix; this also uses $O(n)$ bits.

Finally, each recursive problem has $O(\lg N) = O(L)$ bits of "header" information, containing, e.g., the bounding box of the problem in the top-level coordinate system. Ignoring the header information, the space usage is given by:

$$S(n) = 2\sqrt{n/L}S(\sqrt{nL}) + O(n\sqrt{\lg n}),$$

which after r levels of recursion becomes:

$$S(N) = 2^r \frac{N^{1-1/2^r}}{L^{1-1/2^r}} S(N^{1/2^r} L^{1-1/2^r}) + O\left(2^r N \sqrt{\lg(N^{1/2^r} L^{1-1/2^r})}\right).$$

The recursion is terminated for the first level r where $2^r \geq \lg N / \lg\lg N$. At this level, the problems are of size $O\big((\lg N)^2\big)$ and $\Omega\big((\lg N)^{1.5}\big)$ and the second term in the space usage becomes $O(N \lg N)$ bits. Applying $S(n) = O(n \lg n)$ for the base case, we see that the first term is $O((\lg N / \lg\lg N) \cdot N \cdot \lg\lg N) = O(N \lg N)$ bits, and the space used by the header information is indeed negligible.

3.2 The slab-rank and slab-select problems

The input to each recursive problem of size n is given in local coordinates (i.e. from $[n] \times [n]$). Upon decomposing the query to this problem, we need to solve the following *slab-rank* problem (with a symmetric variant for vertical slabs):

> Given a point $p = (i^*, j^*)$ in top-level coordinates, which is mapped to (i, j) in a recursive problem of size n, such that (i, j) that lies in a horizontal slab of size $n \times k$, map (i, j) to the appropriate position (i', j') in the size k problem represented by this slab.

We formalize the "inverse" *slab-select* problem as follows:

> Given a rectangle R in the coordinate system of a recursive problem, return the top-level coordinates of all points that lie within R.

The following lemma assumes and builds upon the four "global" data structures mentioned after the statement to Theorem 1 (proof omitted):

Lemma 4. *The slab-rank problem can be solved in $O(\lg N/\lg\lg N)$ time, and the slab-select problem in $O(\lg N/\lg\lg N)$ time as well, provided that R contains at most $O\left(\sqrt{\lg N}\right)$ points.*

3.3 Encoding 2-sided queries

We now show that to answer 2-sided range maxima queries at point q ($RMQ(q)$ hereafter), a linear number of bits suffice to encode priority information:

Lemma 5. *Given a set S of n points from $\mathbb{R}^2$ and relative priorities given as a permutation π on $[n]$, the query $RMQ(q)$ can be reduced to point location of q in a collection of at most n horizontal semi-open line segments, whose left endpoints are points from S, and whose right endpoints have x-coordinate equal to the x-coordinate of some point from S. Further, given at most $3n$ bits of extra information, the collection of line segments can be reconstructed from S.*

Proof. Assume the points are in general position and that the 2-sided query is open to the top and left. Associate each point $p = (x(p), y(p)) \in S$ with a horizontal semi-open *line of influence*, possibly of length zero, whose left endpoint (included in the line) is p itself, and is denoted by $Inf(p)$, and contains all points q such that $y(q) = y(p)$, $x(q) \geq x(p)$ and $RMQ(q) = p$. It can be seen that (see e.g. [16]) the answer to $RMQ(q)$ for any $q \in \mathbb{R}^2$ can be obtained by shooting a vertical ray upward from q until the first line $Inf(p)$ is encountered; the answer to $RMQ(q)$ is then p (if no line is encountered then there is no point in the 2-sided region specified by q). See Fig. 3 for an example.

The set $Inf(S) = \{Inf(p) | p \in S\}$ can be computed by sweeping a vertical line from left to right. At any given position $x = t$ of the sweep line, the sweep line will intersect $Inf(S')$ for some set S' (initially $S' = \emptyset$). If $S' = p_{i_1}, \ldots, p_{i_r}$ such that $y(p_{i_1}) > \ldots > y(p_{i_r})$ then it follows that $\pi(i_1) < \ldots < \pi(i_r)$ (the current lines of influence taken from top to bottom represent points with increasing priorities). Upon reaching the next point p_s such that $y(p_{i_j}) < y(p_s) < y(p_{i_{j+1}})$, either (i) $\pi(s) < \pi(i_j)$—in this case $Inf(p_s)$ is empty—or (ii) $\pi(k) > \pi(i_j)$. In the latter case, it may be that $\pi(k) > \pi(i_{j+1}), \ldots \pi(i_{j+k})$ for some $k \geq 0$, which would mean that $Inf(p_{i_{j+1}}), \ldots, Inf(p_{i_{j+k}})$ are terminated, with their right endpoints being $x(p_s)$. To construct $Inf(S)$, therefore, only $O(n)$ bits of information are needed: for each point, one bit is needed to indicate whether case (i) or (ii) holds, and in the latter case, the value of k needs to be stored. However, k can be stored in unary using $k + 1$ bits, and the total value of k, over the course of the entire sweep, is at most n, giving a total of at most $3n$ bits[5] □

[5] A tight bound of $n \lg 5 + o(n) \sim 2.33n$ bits can be shown [11].

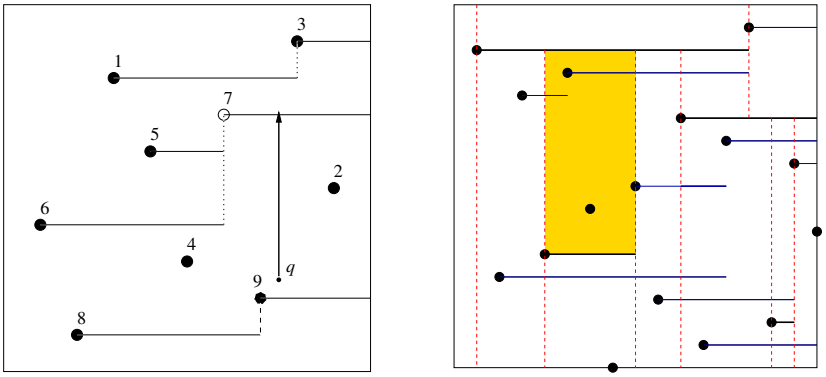

Fig. 3. (Left) Example for Lemma 5. The horizontal lines are the lines of influence. Vertical dotted lines show where a point has terminated the line of influence of another point. The arrow shows how point location in the lines of influence answers the 2-sided query with lower right hand corner at q, returning the point with priority 7. (Right) We select the thick lines of influence, and create a vertical decomposition of all lines of influence shown by vertical dashed lines. One of the regions obtained is highlighted.

3.4 Data structures for 2-sided queries

In this section we show the following lemma. The given 2-sided problem of size n is viewed as a point location problem among $O(n)$ horizontal line segments as in Lemma 5. As the space available is only $O(n\sqrt{\lg n})$ bits, we use an implicit representation of the problem, using and building upon the four "global" data structures mentioned after the statement to Theorem 1.

Lemma 6. *Given a recursive sub-problem of size n, we can answer 2-sided queries on this problem in $O(\lg N)$ time using $O(n\sqrt{\lg n})$ bits of space.*

We begin with an overview. We first create $Inf(T)$, and choose a parameter $\lambda = \sqrt{\lg n}$. We then select a set of points $T' \subseteq T$ with the following properties: (a) $|T'| = O(n/\lambda)$; (b) the *vertical decomposition*, whereby we shoot vertical rays upward and downward from each endpoint of each segment in $Inf(T')$ until they hit another segment, of the plane induced by $Inf(T')$[6] decomposes the plane into $O(n/\lambda)$ rectangular regions each of which has at most $O(\lambda)$ points from T and parts of line segments from $Inf(T)$ in it (see Fig. 3(R)). T' always exists and can be found by plane sweep [3, Section 3],[9, Section 4.3]. We store a standard point location data structure, called the *skeleton*, on T': this requires $O((n/\lambda)\lg n) = O\big(n\sqrt{\lg n}\big)$ bits. We also store $O(\lambda)$ bits of information with each region (including the $O(\lambda)$-bit encoding of priority information from Lemma 5).

Given a query point q, we first perform a point location query on the skeleton to determine the region R in which q lies. We now need to solve the original point location problem within R, and perform a slab-select to determine the points

[6] Note that the extent of a line segment in $Inf(T')$ is defined, as originally, wrt points in T, and not wrt the points in T'.

of T that lie within this region. This, together with the priority information, allows us to partially—but not fully, since lines of influence may originate from outside R—reconstruct the point location problem within R. To handle lines of influence starting outside R, we do a binary search with $O(\lg \lambda)$ steps, each step including a a slab-select, giving the claimed bound. The details are as follows.

Preprocessing. Let R be any region, and let $\mathit{Left}(R)$ ($\mathit{Right}(R)$) be the set of line segments from $\mathit{Inf}(T)$ that intersect the left (right) boundaries of R, and let $P(R)$ be the set of points from T in R. We store the following data for R:

1. For each line segment $\ell \in \mathit{Left}(R)$, ordered top-to down by y-axis, a bit that indicates whether the right endpoint of ℓ is in R or not; similarly for $\ell \in \mathit{Right}(R)$, a bit indicating whether ℓ begins in R or not.
2. If the left boundary of R is adjacent to other regions $R_1, R_2, \ldots$ (taken from top to bottom) and $l_i \geq 0$ represents the number of line segments from $\mathit{Left}(R)$ that also intersect R_i, then we store a bit-string $0^{l_1}10^{l_2}1\ldots$. A similar bit-string is stored for the right boundary of R.
3. For each point in $P(R)$ and each line segment in $\mathit{Left}(R)$, a bit-string of length $|P(R)| + |\mathit{Left}(R)|$ whose i-th bit indicates whether the i-th largest y-coordinate in $P(R) \cup \mathit{Left}(R)$ is from $P(R)$ or $L(R)$.
4. Suppose that a line segment $\ell = \mathit{Inf}(p)$ for some $p \in T$ crosses $m \geq \lambda$ regions. Then, in every λth region that ℓ crosses, we explicitly store the region containing p, and p's local coordinates.
5. Finally, for each point $p \in P(R)$, we store the sequence of bits from Lemma 5, which indicates whether p has a non-empty $\mathit{Inf}(p)$ and if so, for how many lines from $\mathit{Left}(R) \cup \mathit{Inf}(P(R))$, p is a right endpoint (p cannot be a right endpoint of any other line in $\mathit{Inf}(T)$, by the construction of the skeleton).

The purpose of (1) and (2) is to trace a line segment ℓ as it crosses multiple regions: if ℓ crosses from a region R' to a region R'' on its right, then given its position in $\mathit{Right}(R')$, we can deduce its position in $\mathit{Left}(R'')$. Using (4), after tracing $\ell = \mathit{Inf}(p)$ through $\leq \lambda$ regions, we will discover (at least) the region containing p. The skeleton takes $O(n\sqrt{\lg n})$ bits, so we now add up the space used by (1)-(5). By construction, $|\mathit{Left}(R)|$, $|\mathit{Right}(R)|$ and $|P(R)|$, summed over all regions R, is $O(n)$. The space bound for (1) and (3) is therefore $O(n)$ bits. The number of 1s in the bit string of (3), summed over all regions, is $O(n/\lambda)$, as there are $O(n/\lambda)$ regions and the graph which indicates adjacency of regions is planar; the number of 0s is $O(n)$ as before. The space used by (4) is $O(n\sqrt{\lg n})$ bits again, as for every $O(\sqrt{\lg n})$ portions of line segments in the regions we store $O(\lg n)$ bits. Finally, the space used for (5) is $O(n)$ bits by Lemma 5.

Query algorithm. Given a query point q in a sub-problem of size n (assume that we have q's local and top-level coordinates), we answer $RMQ(q)$ as follows:

(a) Do a planar point location in the skeleton, and find a region R in which the point q lies. Perform slab-select on R to get $P(R)$.

(b) As we know how many segments from $Left(R)$ lie vertically between any pair of points in $P(R)$, when we are given the data in (5) above, we are able to determine whether the x-coordinate of a given point p in $P(R)$ is the right endpoint of a line from either $Left(R)$ or $Inf(P(R))$. Thus, we have enough information to determine $Inf(p)$ for all $p \in P(R)$ (at least until the right boundary of R). Furthermore, for each line in $Left(R)$ that terminates in R, we also know (the top-level coordinates of) its right endpoint.

(c) Using the top-level coordinates of q, we determine the nearest segment from $Inf(P(R))$ that is above q.

(d) Using the top-level coordinates of q we also find the set of segments from $Left(R)$ whose right endpoints are not to the left of q. Let this set be $Left^*(R)$. We now determine the nearest segment from $Left^*(R)$ that is above q. Unfortunately, although $|Left^*(R)| = O(\lambda)$, since the segments in $Left^*(R)$ originate in points outside R, we do *not* have their y-coordinates. Hence, we need to perform the following binary search on $Left^*(R)$:

  (d1) Take the line segment $\ell \in Left^*(R)$ with median y-coordinate, and suppose that $\ell = Inf(p)$. The first task is to find the region R_p containing p, as follows. Use (2) to determine which of the adjacent regions of R ℓ intersects, say this is R'. If ℓ ends in R', or $R' = R_p$ and we are done. Otherwise, use (1) to locate ℓ in $Left(R')$ and continue.

  (d2) Once we have found R_p, we perform a slab-select on R' to determine $P(R_p)$, and sort $P(R_p)$ by y-axis. Then we perform (c) above on $P(R_p)$, thus determining which points of $P(R_p)$ have lines of influence that reach the right boundary of R_p. Using this we can now determine the (top-level) coordinates of p.

  (d3) We compare the top-level y-coordinates of p and q and recurse.

(e) We take the lower of the lines found in (d) and (e) and use it to return a candidate. Observe that we have the top-level coordinates of this candidate.

We now derive the time complexity of a 2-sided query. Step (a) takes $O(\lg n)$ for the point location, and $O(\lg N/\lg\lg N)$ for the slab-select. Step (b) can be done in $O(\lg n) = O(\lg N)$ time by running the plane sweep algorithm of Lemma 5 (recall that $|P(R)| = O(\sqrt{\lg n})$—a quadratic algorithm will suffice). Step (c) likewise can be done by a simple plane sweep in $O(\lg n)$ time. Step (d1) is iterated at most $O(\sqrt{\lg n})$ times before R_p is found since every λ-th region intersected by ℓ contains information about p. Each iteration of (d1) takes $O(1)$ time: operations on the bit-strings are done either by table lookup if the bit-string is short ($O(\lambda)$ bits), or else using rank and select operations [19], if the bit string is long (as e.g. the bit-string in (2) may be) – these entirely standard tricks are not described in detail. Step (d2) takes $O(\lg N/\lg\lg N)$ time as before. Steps (d1)-(d3) are performed $O(\lg\lambda) = O(\lg\lg N)$ times, so this takes $O(\lg N)$ time overall. Step (e) is trivial. We have thus shown Lemma 6.

3.5 Putting things together

Section 3.1 shows that our data structure occupies $O(N)$ words. The dominant term in the running time is due to solving $O(\lg\lg N)$ 2-sided queries using

Lemma 6, taking $O(\lg N \lg\lg N)$ time. The $O(\lg\lg N)$ square-aligned queries are solved in $O(1)$ time each. The $O(1)$ problems at the bottom of the recursion are solved in $O((\lg\lg N)^2)$ time. We scan all $O(\lg\lg N)$ candidates to find the answer (any candidates given in local coordinates are converted to top-level coordinates in $O(\lg N/\lg\lg N)$ time each, or $O(\lg N)$ time overall). This proves Theorem 1.

4 Succinct indices for 2-sided queries

We now consider succinct indices for 2-sided range maxima queries over N points in rank space. Our results are stand-alone variants of Lemma 6 and reuse its structure (proofs can be found in [11]). The indices encode the priority information, but not the point coordinates, which are assumed to be accessible in one of two ways. First, we consider the case where points are reported through an orthogonal range reporting query, such that a query that results in k points being reported takes $T(N,k)$ time (assume that $T(N,O(k)) = O(T(N,k))$). Then:

Lemma 7. *Let $\lambda \geq 2$ be some parameter. There is a succinct index of size $O(N+(N\lg N)/\lambda))$ bits such that 2-sided range maxima queries can be answered in $O(\lg N + \lg\lambda(\lambda + T(N,\lambda))$ time.*

In the *permuted-point* model of [4], the point coordinates are stored in read-only memory, and the i-th point (according to an ordering specified by the data structure) can be accessed in $O(1)$ time. We can show:

Lemma 8. *There is a succinct index of $N\lg 5 + o(N) = 2.33N + o(N)$ bits such that 2-sided range maxima queries can be answered in $O(\lg\lg N)$ time in the permuted-point model.*

5 Conclusions

We have introduced a new approach to producing space-efficient data structures for orthogonal range queries, and have applied our approach to give the first linear-space data structure for 2D range maxima that improves upon Chazelle's 1985 linear-space data structure. It would be interesting to try to obtain (say) $O(\lg\lg N)$ running time as in [8] in linear space, or to apply these ideas to related problems such as top-k queries.

References

1. Afshani, P., Arge, L., Larsen, K.D.: Orthogonal range reporting: query lower bounds, optimal structures in 3-D, and higher-dimensional improvements. In: Snoeyink, J., de Berg, M., Mitchell, J.S.B., Rote, G., Teillaud, M. (eds.) Symposium on Computational Geometry. pp. 240–246. ACM (2010)
2. Barbay, J., He, M., Munro, J.I., Rao, S.S.: Succinct indexes for strings, binary relations and multi-labeled trees. In: Bansal, N., Pruhs, K., Stein, C. (eds.) SODA. pp. 680–689. SIAM (2007)

3. Bender, M.A., Cole, R., Raman, R.: Exponential structures for efficient cache-oblivious algorithms. In: Widmayer, P., Ruiz, F.T., Bueno, R.M., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP. LNCS, vol. 2380, pp. 195–207. Springer (2002)
4. Bose, P., Chen, E.Y., He, M., Maheshwari, A., Morin, P.: Succinct geometric indexes supporting point location queries. In: Mathieu, C. (ed.) SODA. pp. 635–644. SIAM (2009)
5. Bose, P., He, M., Maheshwari, A., Morin, P.: Succinct orthogonal range search structures on a grid with applications to text indexing. In: Dehne, F.K.H.A., Gavrilova, M.L., Sack, J.R., Tóth, C.D. (eds.) WADS. LNCS, vol. 5664, pp. 98–109. Springer (2009)
6. Brodal, G.S., Davoodi, P., Rao, S.S.: On space efficient two dimensional range minimum data structures. In: de Berg, M., Meyer, U. (eds.) ESA (2). LNCS, vol. 6347, pp. 171–182. Springer (2010)
7. Brodal, G.S., Jørgensen, A.G.: Data structures for range median queries. In: Dong, Y., Du, D.Z., Ibarra, O.H. (eds.) ISAAC. LNCS, vol. 5878, pp. 822–831. Springer (2009)
8. Chan, T.M., Larsen, K.G., Pătraşcu, M.: Orthogonal range searching on the ram, revisited. In: Proceedings of the 27th annual ACM symposium on Computational geometry. pp. 1–10. SoCG '11, ACM, New York, NY, USA (2011), http://doi.acm.org/10.1145/1998196.1998198
9. Chan, T.M., Patrascu, M.: Transdichotomous results in computational geometry, I: Point location in sublogarithmic time. SIAM J. Comput. 39(2), 703–729 (2009)
10. Chazelle, B.: A functional approach to data structures and its use in multidimensional searching. SIAM J. Comput. 17(3), 427–462 (1988), prel. vers. *FOCS 85*.
11. Farzan, A., Munro, J.I., Raman, R.: Succinct indices for range queries with applications to orthogonal range maxima. Tech. Rep. CS-TR-12-001, U. Leicester (April 2012), available at http://arxiv.org/abs/1204.4835
12. Gabow, H.N., Bentley, J.L., Tarjan, R.E.: Scaling and related techniques for geometry problems. In: Proc. 16th Annual ACM Symposium on Theory of Computing. pp. 135–143. ACM (1984)
13. Golin, M.J., Iacono, J., Krizanc, D., Raman, R., Rao, S.S.: Encoding 2d range maximum queries. In: Asano, T., Nakano, S.I., Okamoto, Y., Watanabe, O. (eds.) ISAAC. LNCS, vol. 7074, pp. 180–189. Springer (2011)
14. JáJá, J., Mortensen, C.W., Shi, Q.: Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In: Fleischer, R., Trippen, G. (eds.) ISAAC. LNCS, vol. 3341, pp. 558–568. Springer (2004)
15. Karpinski, M., Nekrich, Y.: Space efficient multi-dimensional range reporting. In: Ngo, H.Q. (ed.) COCOON. LNCS, vol. 5609, pp. 215–224. Springer (2009)
16. Makris, C., Tsakalidis, A.K.: Algorithms for three-dimensional dominance searching in linear space. Inf. Process. Lett. 66(6), 277–283 (1998)
17. Mehta, D.P., Sahni, S. (eds.): Handbook of Data Structures and Applications. Chapman & Hall/CRC (2009)
18. Nekrich, Y.: Orthogonal range searching in linear and almost-linear space. Comput. Geom. 42(4), 342–351 (2009)
19. Rahman, N., Raman, R.: Rank and select operations on binary strings. In: Kao, M.Y. (ed.) Encyclopedia of Algorithms. Springer (2008)
20. Yuan, H., Atallah, M.J.: Data structures for range minimum queries in multidimensional arrays. In: Charikar, M. (ed.) SODA. pp. 150–160. SIAM (2010)