Succinct Orthogonal Range Search Structures on a Grid with Applications to Text Indexing*

Prosenjit Bose[1], Meng He[2], Anil Maheshwari[1], and Pat Morin[1]

[1] School of Computer Science, Carleton University, Canada, {jit, anil, morin}@cg.scs.carleton.ca
[2] Cheriton School of Computer Science, University of Waterloo, Canada, mhe@uwaterloo.ca

Abstract. We present a succinct representation of a set of n points on an $n \times n$ grid using $n \lg n + o(n \lg n)$ bits[3] to support orthogonal range counting in $O(\lg n / \lg \lg n)$ time, and range reporting in $O(k \lg n / \lg \lg n)$ time, where k is the size of the output. This achieves an improvement on query time by a factor of $\lg \lg n$ upon the previous result of Mäkinen and Navarro [15], while using essentially the information-theoretic minimum space. Our data structure not only can be used as a key component in solutions to the general orthogonal range search problem to save storage cost, but also has applications in text indexing. In particular, we apply it to improve two previous space-efficient text indexes that support substring search [7] and position-restricted substring search [15]. We also use it to extend previous results on succinct representations of sequences of small integers, and to design succinct data structures supporting certain types of orthogonal range query in the plane.

1 Introduction

The two-dimensional *orthogonal range search* problem is a fundamental problem in computational geometry. In this problem, we store a set, N, of points in a data structure so that given a query rectangle R, information about the points in R can be retrieved efficiently. There are two common types of queries: *orthogonal range counting* queries and *orthogonal range reporting* queries. An orthogonal range counting query returns the number of points in $N \cap R$, and an orthogonal range reporting query returns these points. The orthogonal range search problem has applications in many areas of computer science, including databases and computer graphics, and thus has been studied extensively [11, 6, 17, 1, 16]. Many trade-offs for this problem have been achieved. For example, for the two-dimensional range reporting query, there are data structures achieving the optimal $O(\lg n + k)$ query time using $O(n \lg^{\epsilon} n)$ words of space, where k is the size of the output and $0 < \epsilon < 1$ [1], and structures of linear space that

* This work was supported by NSERC of Canada. The work was done when the second author was in School of Computer Science, Carleton University, Canada.

[3] $\lg n$ denotes $\log_2 n$.

answer queries in $O(\lg n + k \lg^{\epsilon} n)$ time [6]. For a recent summary of different results on orthogonal range search, see [16].

In this paper, we mainly study the orthogonal range search problem in two-dimensional *rank space*, i.e. on an $n \times n$ grid, where n is the size of the point set. The general orthogonal range search problem in which the points are real numbers can be reduced to this problem using a standard approach [11]. Thus, solutions to the general range search problem are often based on range search structures in the rank space [11, 1]. For example, one key component for the data structure achieving the optimal $O(\lg n + k)$ query time for orthogonal range reporting by Alstrup *et al.* [1] is a data structure supporting orthogonal range reporting in the rank space in $O(\lg \lg n + k)$ time using $O(n \lg^{\epsilon} n)$ words of space.

More recently, the orthogonal range search problem on an $n \times n$ grid was studied to design *succinct data structures*, and in particular succinct text indexes. Succinct data structures provide solutions to reduce the storage cost of modern applications that process huge amounts of data, such as textual data in databases and on the World Wide Web, geometric data in GIS systems, and genomic data in bioinformatics applications. They were first proposed by Jacobson [14] to encode bit vectors, (unlabeled) trees and planar graphs using space close to the information-theoretic lower bound, while supporting efficient navigation operations in them. For example, Jacobson showed how to represent a tree on n nodes using $2n + o(n)$ bits, so that the parent and the children of a node can be efficiently located. The obvious approach uses $3n$ words, which is about 96 times as much as the space required for the succinct representation on a 64-bit machine. This approach was also successfully applied to various other abstract data types, including dictionaries [18], strings [13, 2, 3], binary relations [2, 3] and labeled trees [12, 9, 2, 3]. For orthogonal range search in rank space, Mäkinen and Navarro [15] designed a succinct data structure that encodes the point set using $n \lg n + o(n \lg n)$ bits to support orthogonal range counting in $O(\lg n)$ time and range reporting in $O(k \lg n)$ time. This space cost is close to the information-theoretic minimum, but their structure requires that there does not exist two points in the set that has the same coordinate in one of the two dimensions.

The above succinct range search structure was further used to design space-efficient text indexes. Mäkinen and Navarro [15] initially designed this structure for the problem of *position-restricted substring search*. The goal is to construct an index for a text string T of length n such that given a query substring P of length m and a range $[i..j]$ of positions in T, the occurrences of P in this range can be reported efficiently. They showed how to reduce the problem to orthogonal range search on an $n \times n$ grid, and designed a text index of $3n \lg n + o(n \lg n)$ bits to support position-restricted substring search in $O(m + \mathtt{occ} \lg n)$ time, where $\mathtt{occ}$ is the number of occurrences of P in T. Chien *et al.* [7] considered the problem of indexing a text string to support the general *substring search* that reports the occurrences of a query pattern P in a text string T. This is a fundamental problem in computer science. They designed a succinct text index using $O(n \lg \sigma)$ bits, where σ is the alphabet size, to support substring search in

$O(m + \lg n(\lg_\sigma n + \texttt{occ} \lg n))$ time. One key data structure in their solution is the same succinct range search structure [15].

1.1 Our Results

In this paper, we design succinct data structures for orthogonal range search on an $n \times n$ grid. Our range search structure is an improvement upon that of Mäkinen and Navarro [15], and we use it to improve previous results on designing space-efficient text indexes for substring search [7] and position-restricted substring search [15]. We also apply our structure to extend the previous result on representing a sequence of small integers succinctly [10], as well as a restricted version of orthogonal range search in which the query range is defined by two points in the point set [4]. More precisely, we present the following results, among which the first one is our main result and the rest are its applications:

1. A succinct data structure that encodes a point set, N, of n points in an $n \times n$ grid using $n \lg n + o(n \lg n)$ bits to support orthogonal range counting in $O(\lg n / \lg \lg n)$ time, and orthogonal range reporting in $O(k \lg n / \lg \lg n)$ time, where k is the size of the output. Compared to the succinct structure of Mäkinen and Navarro [15], this data structure achieves an improvement on query time by a factor of $\lg \lg n$, while still using space close to the information-theoretic minimum. Another improvement is that our structure does not require each point to have a distinct x-coordinate or y-coordinate.
2. A succinct text index of $O(n \lg \sigma)$ bits for a text string T of length n over an alphabet of size σ that supports substring search in $O(m + \lg n(\lg_\sigma n + \texttt{occ} \lg n)/ \lg \lg n)$ time, where m is the length of the query substring, and $\texttt{occ}$ is the number of its occurrences in T. This provides faster query support than the structure of Chien *et al.* [7] while using the same amounts of space.
3. A text index of $3n \lg n + o(n \lg n)$ bits that supports position-restricted substring search in $O(m + \texttt{occ} \lg n / \lg \lg n)$ time. This improves the query time of the index of Mäkinen and Navarro [15] using the same amount of space.
4. A succinct data structure that encodes a sequence, S, of n numbers in $[1..s]$, where $s = \texttt{polylog}(n)$, in $nH_0(S) + o(n)$ bits[4] to support the following query in constant time: given a range, $[p_1..p_2]$, of positions in S and a range, $[v_1..v_2]$, of values, compute the number of entries in $S[p_1..p_2]$ whose values are in the range $[v_1..v_2]$. These entries can also be reported in constant time per entry. This extends the result of Ferragina *et al.* [10] on the same input data to support more operations.
5. A space-efficient data structure that encodes a point set N in the plane in $cn + n \lg n + o(n \lg n)$ bits, where c is the number of bits required to encode the coordinate pair of a point, to provide $O(\lg n / \lg \lg n)$-time support for a restricted version of orthogonal range counting in which the query rectangle is defined by two points in N. The points in the query range can be reported in $O(k \lg n / \lg \lg n)$ time.

All our results are under the word RAM model of $\Theta(\lg n)$-bit word size.

[4] $H_0(S)$ is the zeroth-order empirical entropy of S.

2 Preliminaries

Bit vectors. A key structure for many succinct data structures and for our research is a bit vector $B[1..n]$ that supports `rank` and `select` operations. For $\alpha \in \{0, 1\}$, the operator $\texttt{rank}_B(\alpha, x)$ returns the number of occurrences of α in $B[1..x]$, and $\texttt{select}_B(\alpha, r)$ returns the position of the r^{th} occurrence of α in B. Lemma 1 addresses the problem of succinct representations of bit vectors.

Lemma 1 ([14, 8]). *A bit vector $B[1..n]$ with v 1s can be represented using $n + o(n)$ bits to support the access to each bit,* `rank` *and* `select` *in $O(1)$ time.*

Sequences of small numbers. The rank/select operations can also be performed on a sequence, S, of n integers in $[1..s]$. To define $\texttt{rank}_S(\alpha, x)$ and $\texttt{select}_S(\alpha, r)$, we simply let $\alpha \in \{1, 2, \cdots, s\}$ to extend the definitions of these operations on bit vectors. Ferragina *et al.* [10] proved the following lemma:

Lemma 2 ([10]). *A sequence, S, of n numbers in $[1..s]$, where $2 \leq s \leq \sqrt{n}$, can be represented using $nH_0(S) + O(s(n \lg \lg n)/\log_s n)$ bits to support the access of each number,* `rank` *and* `select` *in $O(1)$ time.*

3 Succinct Range Search Structures on a Grid

In this section, we design a succinct data structure that supports orthogonal range search in rank space. We first design a structure for a narrow grid (more precisely, an $n \times O(\lg^{\epsilon} n)$ grid) in Section 3.1 with the restriction that each point has a distinct x-coordinate. Based on this structure, we further design structures for an $n \times n$ grid in Section 3.2 without any similar restrictions.

3.1 Orthogonal Range Search on an $n \times O(\lg^{\epsilon} n)$ Grid

We first consider range counting on a narrow grid. We make use of the well-known fact that the orthogonal range counting problem can be reduced to *dominance counting* queries. A point whose coordinates are (x_1, y_1) *dominates* another point (x_2, y_2) if $x_1 \geq x_2$ and $y_1 \geq y_2$, and a dominance counting query computes the number of points dominating the query point.

Lemma 3. *Let N be a set of points from the universe $M = [1..n] \times [1..t]$, where $n = |N|$ and $t = O(\lg^{\epsilon} n)$ for any constant ϵ such that $0 < \epsilon < 1$. If each point in N has a distinct x-coordinate, the set N can be represented using $n\lceil \lg t \rceil + o(n)$ bits to support orthogonal range counting in $O(1)$ time.*

Proof. As each point in N has a distinct x-coordinate, we can store the coordinates as a sequence $S[1..n]$, in which $S[i]$ stores the y-coordinate of the point whose x-coordinate is i. Thus S occupies $n\lceil \lg t \rceil$ bits, and it suffices to show how to construct auxiliary data structures of $o(n)$ bits to support dominance counting (recall that dominance counting can be used to support range counting).

We first partition the universe M into regions called *blocks* of size $\lceil \lg^2 n \rceil \times t$ by dividing the first dimension into ranges of size $\lceil \lg^2 n \rceil$. More precisely, the i^{th} block of M under this partition is $L_i = [(i-1)\lceil \lg^2 n \rceil + 1..i\lceil \lg^2 n \rceil] \times [1..t]$. We assume that n is divisible by $\lceil \lg^2 n \rceil$ for simplicity.

For each block L_i, we further partition it into *subblocks* of size $\lceil \lg^\lambda n \rceil \times t$ by dividing the first dimension into ranges of size $\lceil \lg^\lambda n \rceil$, where λ is a constant such that $\epsilon < \lambda < 1$. Under this partition, the j^{th} subblock of L_i is $L_{i,j} = [(i-1)\lceil \lg^2 n \rceil + (j-1)\lceil \lg^\lambda n \rceil + 1..(i-1)\lceil \lg^2 n \rceil + j\lceil \lg^\lambda n \rceil] \times [1..t]$. For simplicity, we assume that $\lceil \lg^2 n \rceil$ is divisible by $\lceil \lg^\lambda n \rceil$.

We construct the following auxiliary data structures:

- A two-dimensional array $A[1..n/\lceil \lg^2 n \rceil, 1..t]$, in which $A[i,j]$ stores the number of points in N dominating the coordinate pair $(i\lceil \lg^2 n \rceil, j)$;
- A two-dimensional array $B[1..n/\lceil \lg^\lambda n \rceil, 1..t]$, in which $B[i,j]$ stores the number of points in N dominating the coordinate pair $(i\lceil \lg^\lambda n \rceil, j)$ in the block that contains this coordinate pair;
- A table C that stores for each possible set of $\lceil \lg^\lambda n \rceil$ points in the universe $[1..\lceil \lg^\lambda n \rceil] \times [1..t]$ (each point in this set has a distinct x-coordinate), every integer i in $[1..\lceil \lg^\lambda n \rceil]$ and every integer j in $[1..t]$, the number of points in this set that dominates the coordinate pair.

We now analyze the space costs of the above data structures. A occupies $n/\lceil \lg^2 n \rceil \times t \times \lceil \lg n \rceil = O(n/\lg^{1-\epsilon} n) = o(n)$ bits. As there are $\lceil \lg^2 n \rceil$ points inside each block, each entry of B can be stored in $O(\lg \lg n)$ bits. Therefore, B occupies $n/\lceil \lg^\lambda n \rceil \times t \times O(\lg \lg n) = O(n \lg \lg n/\lg^{\lambda-\epsilon} n) = o(n)$ bits. To compute the space cost of C, we first count the number, b, of possible $\lceil \lg^\lambda n \rceil$-point set in the universe $[1..\lceil \lg^\lambda n \rceil] \times [1..t]$, where each point in this set has a distinct x-coordinate. We have $b = t^{\lceil \lg^\lambda n \rceil} = 2^{\lceil \lg^\lambda n \rceil \lg t}$. Let $f = \lceil \lg^\lambda n \rceil \lg t$. Then $f = O(\lg^\lambda n \lg \lg n) = o(\lg n)$. By the definition of order notation, there exists a constant n_0 such that $f < \frac{1}{2} \lg n$ for any $n > n_0$. As $b = 2^f$, we have $b < 2^{\frac{1}{2}\lg n} = \sqrt{n}$ when $n > n_0$. Therefore, when $n > n_0$, the space cost of C in bits is less than $\sqrt{n} \times \lceil \lg^\lambda n \rceil \times t$. Thus, the space cost of C is $O(\sqrt{n} \lg^{\lambda+\epsilon} n) = o(n)$ bits. Therefore, the auxiliary data structures occupy $O(n \lg \lg n/\lg^{\lambda-\epsilon} n) = o(n)$ bits in total.

With the above data structures, we can support dominance counting. Let (u, v) be the coordinates of the query point q. Let L_i and $L_{i,j}$ be the block and subblock that contain q, respectively. The result is the sum of the following three values: k_1, the number of points in blocks $L_{i+1}, L_{i+2}, \cdots$ that dominate q; k_2, the number of points in subblocks $L_{i,j+1}, L_{i,j+2}, \cdots, L_{i,v}$ that dominate q, where v is the number of subblocks in block L_i; and k_3, the number of points in subblock $L_{i,j}$ that dominate q. By the definitions of the data structures we constructed, we have $k_1 = A[i, y]$ and $k_2 = B[i \times \lceil \lg^2 n \rceil / \lceil \lg^\lambda n \rceil + j, y]$. To compute k_3, we first compute the coordinates of q inside block $L_{i,j}$ by treating $L_{i,j}$ as a universe of size $\lceil \lg^\lambda n \rceil \times t$, and get the encoding of the subsequence of S that corresponds to points inside $L_{i,j}$. With these we can perform table lookup on C to compute k_3 in constant time. □

We next show how to support range reporting.

Lemma 4. *Let N be a set of points from the universe $M = [1..n] \times [1..t]$, where $n = |N|$ and $t = O(\lg^{\epsilon} n)$ for any constant ϵ such that $0 < \epsilon < 1$. If each point in N has a distinct x-coordinate, the set N can be represented using $n\lceil \lg t \rceil + o(n)$ bits to support orthogonal range reporting in $O(k)$ time, where k is the size of the output.*

Proof. As with the proof of Lemma 3, we encode N as the string S, and divide M into blocks and subblocks. Based on this, we design auxiliary data structures to support orthogonal range reporting. We answer a query in two steps. Given a query rectangle R, we first compute the set, Y, of y-coordinates of the output in $O(k')$ time, where k' is the number of distinct y-coordinates of the points in the output. Then, for each y-coordinate, v, in Y, we compute the points in R whose y-coordinate is v (we spend $O(1)$ time on each such point).

We first show how to compute Y in $O(k')$ time. We construct the following auxiliary data structures:

- A two-dimensional array $D[1..n/\lceil \lg^2 n \rceil, 1..\lceil \lg n \rceil]$. Each entry, $D[i, j]$, stores a bit vector of length t whose l^{th} bit is 1 iff there is at least one point (from the set N) in blocks $L_i, L_{i+1}, \cdots, L_{i+2^j-1}$ whose y-coordinate is l;
- A two-dimensional array $E_i[1..v][1..\lceil \lg v \rceil]$ for each block L_i, where $v = \lceil \lg^2 n \rceil / \lceil \lg^{\lambda} n \rceil$ (i.e. the maximum number of subblocks in a given block). Each entry, $E_i[j, u]$, stores a bit vector of length t whose l^{th} bit is 1 iff there is at least one point (from the set N) in subblocks $L_{i,j}, L_{i,j+1}, \cdots, L_{i,j+2^u-1}$ whose y-coordinate is l;
- A table F which stores for every possible set of $\lceil \lg^{\lambda} n \rceil$ point in the universe $[1..\lceil \lg^{\lambda} n \rceil] \times [1..t]$ (each point in this set has a distinct x-coordinate), every pair of integers i and j in $[1..\lceil \lg^{\lambda} n \rceil]$, a bit vector of length t whose l^{th} bit is 1 iff there is at least one point from this set whose x-coordinate is between (and including) i and j and whose y-coordinate is l.

To analyze the space cost, we have that D occupies $O(n/\lg^2 n \times \lg n \times t) = O(n/\lg^{1-\epsilon} n)$ bits. As there are $n/\lceil \lg^{\lambda} n \rceil$ subblocks in total, all the E_i's occupy $O(n/\lg^{\lambda} n \times \lg \lg n \times \lg^{\epsilon} n) = O(n \lg \lg n / \lg^{\lambda-\epsilon})$ bits. Similarly to the analysis in the proof of Lemma 3, we have F occupies $O(\sqrt{n} \times \lceil \lg^{\lambda} n \rceil \times \lceil \lg^{\lambda} n \rceil \times \lceil \lg^{\epsilon} n \rceil)$ bits. Therefore, these data structures occupy $O(n \lg \lg n / \lg^{\lambda-\epsilon}) = o(n)$ bits.

To use the above data structures to compute Y, let $R = [x_1..x_2] \times [y_1..y_2]$ be the query rectangle. We first show how to compute a bit vector Z of length t, where $Z[i] = 1$ iff there is a point from N whose y-coordinate is i and whose x-coordinates are between (and including) x_1 and x_2. Let $L_{a,b}$ and $L_{c,d}$ be the two subblocks whose ranges of x-coordinates contain x_1 and x_2, respectively. Assume that $a < c$ (the case in which $a = c$ can be handled similarly). Then Z is the result of bitwise OR operation on the the following five bit vectors of length t:

- Z_1, where $Z_1[i] = 1$ iff there is a point in blocks $L_{a+1}, L_{a+2}, \cdots, L_{c-1}$ whose y-coordinate is i;

- Z_2, where $Z_2[i] = 1$ iff there is a point in subblocks $L_{a,b+1}, L_{a,b+2}, \cdots, L_{a,q}$ whose y-coordinate is i (let $L_{a,q}$ be the last subblock in block L_a);
- Z_3, where $Z_3[i] = 1$ iff there is a point in subblocks $L_{c,1}, L_{c,2}, \cdots, L_{c,d-1}$ whose y-coordinate is i;
- Z_4, where $Z_4[i] = 1$ iff there is a point in subblock $L_{a,b}$ that is in the query rectangle and whose y-coordinate is i;
- Z_5, where $Z_4[i] = 1$ iff there is a point in subblock $L_{a,b}$ that is in the query rectangle and whose y-coordinate is i;

To compute Z_1, we first observe that the corresponding range of indexes of blocks is $[a+1..c-1] = [a+1, a+2^g] \cup [c-2^g, c-1]$, where $g = \lfloor \lg(c-a-2) \rfloor$ (similar ideas were used by Bender and Farach-Colton to support range minimum queries [5]). Hence Z_1 is result of bitwise OR operation on the bit vectors stored in $D[a+1, g]$ and $D[c-2^g, g]$. Z_2 and Z_3 can be computed in a similar way using E_a and E_c. Z_4 and Z_5 can be computed by performing table lookups on F in constant time. Therefore, Z can be computed in constant time.

To compute Y using Z in $O(k')$ time, it suffices to provide `rank` and `select` operations on Y in constant time. As Y is of size $t = O(\lg^\epsilon n)$, this can be achieved by precomputing a table of $o(n)$ bits [8].

To further report the points in R, we observe that we store the coordinates as a sequence, S, of numbers in $[1..t]$. The data structures of Ferragina *et al.* [10] designed for Lemma 2 has two parts: a compressed encoding of the sequence of $nH_0(S)$ bits and an auxiliary data structure of $O(s(n \lg \lg n)/\lg_s n)$ bits. Their data structures still work if we replace the first part by the uncompressed version of the original sequence. Thus we can construct the auxiliary data structures in Lemma 2 to support `rank` and `select` on S in constant time. As $t = O(\lg^\epsilon n)$, these data structures occupy $O(n(\lg \lg n)^2/\lg^{1-\epsilon} n) = o(n)$ bits. For each y-coordinate, v, in Y, the set of the points in R whose y-coordinates are equal to v can be computed by performing `rank` and `select` operations on S, which takes constant time per point in the output. □

As Lemma 3 and Lemma 4 both encode and store the coordinates in the same sequence and build auxiliary structures of $o(n)$ bits, we can combine them:

Lemma 5. *Let N be a set of points from the universe $M = [1..n] \times [1..t]$, where $n = |N|$ and $t = O(\lg^\epsilon n)$ for any constant ϵ such that $0 < \epsilon < 1$. If each point in N has a distinct x-coordinate, the set N can be represented using $n\lceil \lg t \rceil + o(n)$ bits to support orthogonal range counting in $O(1)$ time, and to support orthogonal range reporting in $O(k)$ time, where k is the size of the output.*

3.2 Orthogonal Range Search on an $n \times n$ Grid

To design succinct data structures for range search in rank space, we first consider range counting with the restriction that each point has a distinct x-coordinate.

Lemma 6. *Let N be a set of points from the universe $M = [1..n] \times [1..n]$, where $n = |N|$. If each point in N has a distinct x-coordinate, the set N can be represented using $n \lg n + o(n \lg n)$ bits to support orthogonal range counting in $O(\lg n/\lg \lg n)$ time.*

Proof. The main idea is to combine the techniques of Lemma 3 with the generalized wavelet tree structures proposed by Ferragina *et al.* [10] to design a representation of N, based on which we design algorithms to support range counting.

We construct our structure recursively; at each level, we construct an orthogonal range counting structure over a set of points whose y-coordinates are in the range $[1..t]$ using Lemma 3, where $t = O(\lg^{\epsilon} n)$ for any constant ϵ such that $0 < \epsilon < 1$. At the first (i.e. top) level, we consider a conceptual point set N_1 from the universe $M_1 = [1..n] \times [1..t]$. N_1 can be obtained by dividing the range of y-coordinates in the universe M into t ranges of the same size, and there is a point (a, b) in N_1 iff there is a point in N whose x-coordinate is a, and whose y-coordinate is in the i^{th} range. More precisely, if there is a point (x, y) in N, then there is a point $(x, \lfloor y/(n/t) \rfloor)$ in N_1. We then construct an orthogonal range counting structure, C_1 for N_1 using Lemma 3. Note that when we use the approach of Lemma 3 to construct C_1, we store the set N_1 as a sequence S_1 over alphabet $[t]$ in which $S_1[i]$ stores the y-coordinate of the point whose x-coordinate is i. The approach of Lemma 2 can be used here to construct an auxiliary structure of $o(n)$ bits to support rank/select operations on S_1 in constant time. The space cost of the data structures constructed for the first level is clearly $n\lceil \lg t \rceil + o(n)$ bits.

At the second level, we consider t conceptual point sets $N_{2,1}, N_{2,2}, \cdots, N_{2,t}$. The set $N_{2,i}$ is from the universe $M_{2,i} = [1..n_{2,i}] \times [1..t]$, which corresponds to the i^{th} range of y-coordinates of M for the level above (i.e. the first level), where $n_{2,i}$ is the number of points in N whose y-coordinates are in this range. We further divide this range into t subranges of the same size, such that the point (x, y) is in $N_{2,i}$ iff there is a point in N_1 whose x-coordinate is $\texttt{select}_i(S_1, x)$, and whose y-coordinate is in the y^{th} subrange. Note that $\sum_{i=1}^{t} n_{2,i} = n$. Thus, if we combine all the universes $M_{2,1}, M_{2,2}, \cdots, M_{2,t}$ such that the universe $M_{2,i-1}$ is adjacent and to the left of $M_{2,i}$, we can get a universe $M_2 = [1..n] \times [1..t]$. We also transform the coordinates of the points in $N_{2,1}, N_{2,2}, \cdots, N_{2,t}$ into coordinates in the universe M_2, and denote the set that contains all these (n) points N_2. We construct an orthogonal range counting structure, C_2 for N_2 using Lemma 3 (Same as S_1, S_2 denotes the corresponding string). We can count the total number of points in the sets $N_{2,1}, N_{2,2}, \cdots, N_{2,j-1}$ in constant time for any given j by performing range counting on C_1. Thus, we can determine the range of x-coordinates in C_2 that correspond to the points in $C_{2,i}$ in constant time, which allows us to use C_2 to answer orthogonal range queries on each set $C_{2,i}$ in constant time. We also construct the auxiliary structures of Lemma 2 to support rank/select operations on C_2 in constant time, which can be further used to support rank/select operations on each substring of S_2 that corresponds to the set $C_{2,i}$. The total space of the data structures constructed for the second level is thus $n\lceil \lg t \rceil + o(n)$ bits.

We continue the above process recursively, and at each level l, we construct t point sets for each point set considered at level $l - 1$. Figure 1 illustrates the hierarchy of our structure. This structures use $n\lceil \lg t \rceil + o(n)$ bits for each level

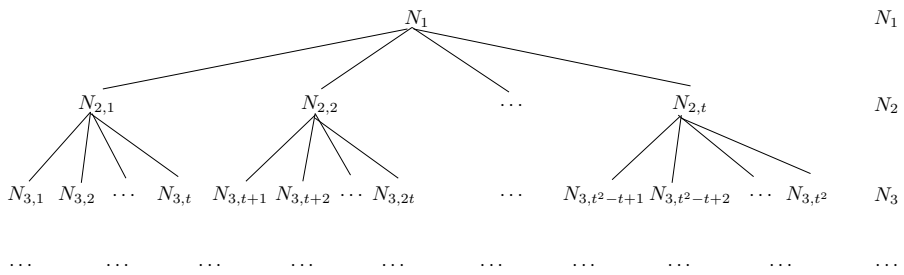

Fig. 1. The hierarchy of the data structures in Lemma 6.

to support orthogonal range counting in each point set at this level, as well as rank/select operations on the substrings of S_l corresponding to each set. Note that the substring of S_l and the sub-universe of M_l that correspond to any set at this level can be located in a top-down traversal, performing range counting at each level in constant time until we reach level l. We continue this process until we can no longer divide a point set into t subsets (i.e. the y-coordinates of the points in this set are from a range of size t in M). Thus our structures have $\log_t n$ levels, and the set of data structures for each level occupy $n\lceil \lg t \rceil + o(n)$ bits. Therefore, the overall space of our structures is $n \lg n + o(n \lg n)$ bits.

We now design a recursive algorithm to support orthogonal range counting using our data structures. Let $R = [x_1..x_2] \times [y_1..y_2]$ be the query rectangle. We consider the case in which $y_2 - y_1 \geq n/t$. Let $z_1 = \lceil y_1/(n/t) \rceil$ and $z_2 = \lfloor y_2/(n/t) \rfloor$. Then R can be partitioned into three query rectangles: $R_1 = [x_1..x_2] \times [y_1..z_1 n/t]$, $R_2 = [x_1..x_2] \times [z_1 n/t + 1..z_2 n/t]$ and $R_3 = [x_1..x_2] \times [z_2 n/t + 1..y_2]$. The result is the sum of the numbers, r_1, r_2 and r_3, of points in R_1, R_2 and R_3, respectively. We observe that r_2 can be computed by performing an orthogonal range counting query over the structure C_1, using $[x_1..x_2] \times [z_1..z_2 - 1]$ as the query rectangle. Thus we need only compute r_1 (the computation of r_3 is similar). Note that R_2 is the maximum sub-rectangle of R whose range of y-coordinates starts with a multiple of n/t and whose width is divisible by n/t, and we use C_1 to compute the number of points in it. Using the same strategy, we can compute the maximum sub-rectangle of R_1 whose range of y-coordinates starts with a multiple of n/t^2 and whose width is divisible by n/t^2, and we use C_{2,z_1-1} to compute this result. To perform this query on C_{2,z_1-1}, we need to scale down the range of x-coordinates of R_1 to $[\texttt{rank}_{S_1}(z-1, x_1)..\texttt{rank}_{S_1}(z-1, x_2)]$. The number of the points in the remaining part of R_1 (note that they are all above this sub-rectangle) can be computed in a recursive fashion using the same approach. Thus r_1 can be computed by performing a top-down traversal to at most the bottom level, and we require constant time per level. Therefore, r_1 can be computed in $O(\log_t n) = O(\lg n / \lg \lg n)$ time. Hence we can compute r in $O(\lg n / \lg \lg n)$ time. The case in which $y_2 - y_1 < n/t$ can be handled similarly. □

We now remove the restriction that each point has a distinct x-coordinate:

Lemma 7. *Let N be a set of points from the universe $M = [1..n] \times [1..n]$, where $n = |N|$. N can be represented using $n \lg n + o(n \lg n)$ bits to support orthogonal range counting in $O(\lg n / \lg \lg n)$ time.*

Proof. We construct a point set N' in which each point has a distinct x-coordinate as follows: Sort the points in N in increasing order using their x-coordinates as the primary key and their y-coordinates as the secondary key. If the i^{th} point in this order has y-coordinate y_i, we add the point (i, y_i) into N'. We also construct a bit vector C to encode the number of points having the same x-coordinates. More precisely, $C = 10^{k_1}10^{k_2} \cdots 10^{k_n}$, where k_j is the number of points whose x-coordinates are j. See Figure 2 for an example.

We represent N' using Lemma 6 using $n \lg n + o(n \lg n)$ bits. There are n 1s and n 0s in C, so we can represent C in $2n + o(n)$ bits to support rank/select operations. The overall space cost of our data structures is $n \lg n + o(n \lg n)$ bits.

To answer an orthogonal range query, let $R = [x_1..x_2] \times [y_1..y_2]$ be the query rectangle. Consider the points from N that is in R. We observe that the points in N' corresponding to them are in the rectangle $R' = [\texttt{rank}_C(0, \texttt{select}_C(1, x_1)) + 1..\texttt{rank}_C(0, \texttt{select}_C(1, x_2+1))] \times [y_1..y_2]$. Thus we need only perform an orthogonal range counting query on N' using R' as the query rectangle. □

Lemma 8. *Let N be a set of points from the universe $M = [1..n] \times [1..n]$, where $n = |N|$. N can be represented using $n \lg n + o(n \lg n)$ bits to support orthogonal range reporting in $O(k \lg n / \lg \lg n)$ time, where k is the size of the output.*

Proof. We only consider the case in which each point has a distinct x-coordinate; the approach in Lemma 7 can be used to extend this to the more general case.

We use the approach of Lemma 6 to construct a hierarchy of structures. The only difference is that at each level i, we use Lemma 4 to construct C_i to support range reporting on N_i. The same algorithm can be used to support orthogonal range reporting; at each level we report a set of points in the answer. The challenge here is how to get the original coordinates of each point reported at the i^{th} level. Let (x, y) be the coordinates of a point, v, reported in the set $N_{j,k}$. Then the set $N_{j-1,\lceil k/t \rceil}$ contains v in the level above. Note that in previous steps, we have computed the number, u, of points in the sets $N_{j-1,1}, N_{j-1,2}, \cdots, N_{j-1,\lceil k/t \rceil - 1}$. The x-coordinate of the point in $N_{j,k}$ corresponding to v is $\texttt{select}_k(S_{j-1}, x + \texttt{rank}_k(S_{j-1}, u)) - u)$. Using this approach, we can go up one level at a time, until we reach the top level, where we get the original x-coordinate of v. Thus the x-coordinate of v can be computed in $O(\lg n / \lg \lg n)$ time. To retrieve the y-coordinate of v, we use the fact that each successive level (after level i) divides the range in N corresponding to each y-coordinate at the level above into t ranges of the same size. Thus, by going down our hierarchy of structures until reaching the bottom level, we can compute the original y-coordinate of v. This can be performed in $O(\lg n / \lg \lg n)$ time. □

Combing Lemma 7 and Lemma 8, we have our main result:

Theorem 1. *Let N be a set of points from the universe $M = [1..n] \times [1..n]$, where $n = |N|$. N can be represented using $n \lg n + o(n \lg n)$ bits to support*

orthogonal range counting in $O(\lg n/\lg\lg n)$ time, and orthogonal range reporting in $O(k\lg n/\lg\lg n)$ time, where k is the size of the output.

4 Applications

Substring search. The succinct text index of Chien *et al.* [7] uses the succinct orthogonal range search structure of Mäkinen and Navarro [15]. Thus, we can speed up substring search by using our structure in Theorem 1:

Theorem 2. *A text string T of length n over an alphabet of size σ can be encoded in $O(n\lg\sigma)$ bits to support substring search in $O(m + \lg n(\lg_\sigma n + \texttt{occ}\lg n)/\lg\lg n)$ time, where m is the length of the query substring, and* `occ` *is the number of its occurrences in T.*

Position-restricted substring search. As Mäkinen and Navarro [15] designed a text index that supports position-restricted substring search by reducing this problem to orthogonal range search on a grid, we can improve their result by applying Theorem 1:

Theorem 3. *Given a text string T of length n over an alphabet of size σ, there is an index of $O(3n\lg n)$ bits that supports position-restricted range search in $O(m+\texttt{occ}(\lg n)/\lg\lg n)$ time, where m is the length of the query substring, and* `occ` *is the number of its occurrences in T.*

Sequences of small numbers. Lemma 2 is interesting only if $s = o(\lg n/\lg\lg n)$ because otherwise, the second term in its space bound becomes a dominating term. Thus, Ferragina *et al.* [10] designed another approach to encode a sequence, S, of n integers bounded by $s = \texttt{polylog}(n)$ in $nH_0(S) + o(n)$ bits to support rank/select operations in constant time. We can further extend their representation to support one more operation: retrieving the entries in any given subsequence of S whose values are in a given range. This is equivalent to the problem of supporting range search on an $n \times t$ grid where $t = \texttt{polylog}(n)$ (each point has a distinct x-coordinate). If we apply the techniques in Section 3.2 to this problem, we only build a constant number of levels of structures. The approach in [10] can also be applied here to achieve compression. Thus:

Theorem 4. *A sequence, S, of n numbers in $[1..s]$, where $s = \texttt{polylog}(n)$, can be encoded in $nH_0(S) + o(n)$ bits such that given a range, $[p_1..p_2]$, of positions in S and a range, $[v_1..v_2]$, of values, the number of entries in $S[p_1..p_2]$ whose values are in the range $[v_1..v_2]$ can be computed in constant time. These entries can be listed in $O(k)$ time, where k is the size of the output. The access to each number,* `rank` *and* `select` *operations can also be supported in $O(1)$ time.*

A restricted version of orthogonal range search. We consider a restricted version of range search (a weaker operation was proposed by Bauernöppel *et al.* [4]) and we have the following theorem (see Appendix B for the proof):

Theorem 5. *A point set N in the plane can be encoded in $cn + n \lg n + o(n \lg n)$ bits, where $n = |N|$ and c is the number of bits required to encode the coordinate pair of each point, to support orthogonal range counting in $O(\lg n / \lg \lg n)$ time and orthogonal range reporting in $O(k \lg n / \lg \lg n)$ time (k is the size of the output) if the query rectangle is defined by two points in N.*

References

1. S. Alstrup, G. S. Brodal, and T. Rauhe. New data structures for orthogonal range searching. In *FOCS*, pages 198–207, 2000.
2. J. Barbay, A. Golynski, J. I. Munro, and S. S. Rao. Adaptive searching in succinctly encoded binary relations and tree-structured documents. *Theoretical Computer Science*, 387(3):284–297, 2007.
3. J. Barbay, M. He, J. I. Munro, and S. S. Rao. Succinct indexes for strings, binary relations and multi-labeled trees. In *SODA*, pages 680–689, 2007.
4. F. Bauernöppel, E. Kranakis, D. Krizanc, A. Maheshwari, J.-R. Sack, and J. Urrutia. Planar stage graphs: Characterizations and applications. *Theoretical Computer Science*, 175(2):239–255, 1997.
5. M. A. Bender and M. Farach-Colton. The LCA problem revisited. In *LATIN*, pages 88–94, 2000.
6. B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal on Computing*, 17(3):427–462, 1988.
7. Y.-F. Chien, W.-K. Hon, R. Shah, and J. S. Vitter. Geometric burrows-wheeler transform: Linking range searching and text indexing. In *DCC*, pages 252–261, 2008.
8. D. R. Clark and J. I. Munro. Efficient suffix trees on secondary storage. In *SODA*, pages 383–391, 1996.
9. P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan. Structuring labeled trees for optimal succinctness, and beyond. In *FOCS*, pages 184–196, 2005.
10. P. Ferragina, G. Manzini, V. Mäkinen, and G. Navarro. Compressed representations of sequences and full-text indexes. *ACM Trans. Alg.*, 3(2):20, 2007.
11. H. N. Gabow, J. L. Bentley, and R. E. Tarjan. Scaling and related techniques for geometry problems. In *STOC*, pages 135–143, 1984.
12. R. F. Geary, R. Raman, and V. Raman. Succinct ordinal trees with level-ancestor queries. *ACM Transactions on Algorithms*, 2(4):510–534, 2006.
13. R. Grossi, A. Gupta, and J. S. Vitter. High-order entropy-compressed text indexes. In *SODA*, pages 841–850, 2003.
14. G. Jacobson. Space-efficient static trees and graphs. In *FOCS*, pages 549–554, 1989.
15. V. Mäkinen and G. Navarro. Rank and select revisited and extended. *Theor. Comput. Sci.*, 387(3):332–347, 2007.
16. Y. Nekrich. Orthogonal range searching in linear and almost-linear space. *Computational Geometry: Theory and Applications*, 42(4):342–351, 2009.
17. M. H. Overmars. Efficient data structures for range searching on a grid. *Journal of Algorithms*, 9(2):254–275, 1988.
18. R. Raman, V. Raman, and S. R. Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Transactions on Algorithms*, 3(4):43, 2007.

Appendix

A An example in the proof of Lemma 7.

Figure 2 gives an example of the point set N' and bit vector C constructed from N in the proof of Lemma 7.

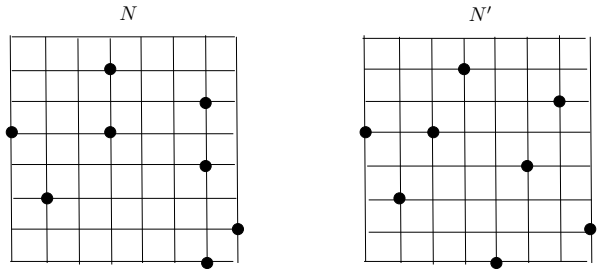

Fig. 2. An example in the proof of Lemma 7.

B Proof of Theorem 5

We first sort the coordinate pairs of the points in N in lexicographic order, and store them in a sequence S. S occupies cn bits. We next construct a point set, N^*, in an $n \times n$ grid as follows. Let p be a point in N and let i be the index of p in lexicographic order. There are n points in N, so there are at most n distinct y-coordinates. Assume that the y-coordinate of p is the j^{th} smallest among these distinct values. We then add a point (i, j) into the set N^*.

We construct an orthogonal range search structure, W, for the point set N^* using Theorem 1. To answer a query, let q and r be two query points in N, and let a and b be the indexes of the coordinate pairs of q and r in S, respectively. Then a and b are the x-coordinates of the two points, s and t, in N^* that correspond to q and r, respectively. The corresponding y-coordinates of s and t can be computed using S in $O(\lg n / \lg \lg n)$ time. The coordinates of s and t define a rectangle, R, in the grid, and we use it as a query rectangle to perform orthogonal range search on N^*. The number of points in N^* that is inside R is equal to the number of points in N that is inside the rectangle defined by q and r, which can be computed in $O(\lg n / \lg \lg n)$ time. The coordinates of the points inside R in the grid can be computed in $O(k \lg n / \lg \lg n)$ time, and we simply use their x-coordinates to index into S to retrieve the coordinates of the corresponding points in N. □