Succinct Color Searching in One Dimension*

Hicham El-Zein[1], J. Ian Munro[2], and Yakov Nekrich[3]

1 **Cheriton School of Computer Science, University of Waterloo, Ontario, Canada**
helzein@uwaterloo.ca
2 **Cheriton School of Computer Science, University of Waterloo, Ontario, Canada**
imunro@uwaterloo.ca
3 **Cheriton School of Computer Science, University of Waterloo, Ontario, Canada**
ynekrich@uwaterloo.ca

Abstract

In this paper we study succinct data structures for one-dimensional color reporting and color counting problems. We are given a set of n points with integer coordinates in the range $[1, m]$ and every point is assigned a color from the set $\{1, \ldots, \sigma\}$. A color reporting query asks for the list of distinct colors that occur in a query interval $[a, b]$ and a color counting query asks for the number of distinct colors in $[a, b]$.

We describe a succinct data structure that answers approximate color counting queries in $O(1)$ time and uses $\mathcal{B}(n, m) + O(n) + o(\mathcal{B}(n, m))$ bits, where $\mathcal{B}(n, m)$ is the minimum number of bits required to represent an arbitrary set of size n from a universe of m elements. Thus we show, somewhat counterintuitively, that it is not necessary to store colors of points in order to answer approximate color counting queries. In the special case when points are in the rank space (i.e., when $n = m$), our data structure needs only $O(n)$ bits. Also, we show that $\Omega(n)$ bits are necessary in that case.

Then we turn to succinct data structures for color reporting. We describe a data structure that uses $\mathcal{B}(n, m) + nH_d(S) + o(\mathcal{B}(n, m)) + o(n \lg \sigma)$ bits and answers queries in $O(k+1)$ time, where k is the number of colors in the answer, and $nH_d(S)$ ($d = \log_\sigma n$) is the d-th order empirical entropy of the color sequence. Finally, we consider succinct color reporting under restricted updates. Our dynamic data structure uses $nH_d(S) + o(n \lg \sigma)$ bits and supports queries in $O(k + 1)$ time.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Succinct Data Structures, Range Searching, Computational Geometry

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.30

1 Introduction and Motivation

Range search problems are problems where a point set is preprocessed so that certain information about a query region can be efficiently computed. These problems are of fundamental importance in computational geometry, both in the study of their optimality with respect to space and query time, and as tools employed to provide efficient solutions to various geometric problems. In this paper we focus on the following two problems. One dimensional color range reporting (counting): Given a set of colored points $\mathcal{P}$, preprocess $\mathcal{P}$ into an efficient data structure so that for any range $\mathcal{Q} = [a, b]$ the distinct colors contained in $\mathcal{P} \cap \mathcal{Q}$ can be reported (counted).

We study both problems in the context of succinctness, where the goal is to achieve the optimal space requirement plus a lower order term, while maintaining fast query time.

* This work was sponsored by the NSERC of Canada and the Canada Research Chairs Program.

Designing succinct data structures is an area of interest in theory and practice motivated by the need of storing large amount of data using the smallest space possible. In recent years there has been a surge of interest in succinct data structures for computational geometry [5, 8, 15]. For further reading and more in-depth coverage of succinct data structures we refer the reader to the survey by Munro and Rao [16].

Previous Work. If the input points are in the rank space, one-dimensional color reporting queries can be answered in $O(k+1)$ time using $nH_d(S) + o(n)\lg\sigma + O(n\lg\lg\sigma)$ bits [2, 4, 6], where $d = o(\log_\sigma n)$ and $H_d(S)$ is the *d-th* order empirical entropy of the given sequence of colors S. In the general case, one-dimensional color reporting queries can be answered in $O(\lg n + k)$ time in the static and dynamic scenarios as shown by Janardan and Lopez [17] and Gupta et al. [26]. Muthukrishnan [20] later described a static $O(n)$ space data structure that answers queries in $O(k+1)$ time when all point coordinates are bounded by n. His result implies an $O(n)$-words data structure that answer queries in $O(\min(\lg\lg m, \sqrt{\lg n/\lg\lg n}) + k)$ time using the reduction-to-rank-space technique, where $O(\min(\lg\lg m, \sqrt{\lg n/\lg\lg n}))$ is the time needed to answer a predecessor query [27, 10]. A dynamic data structure of Mortensen [18] supports queries and updates in $O(\lg\lg n + k)$ and $O(\lg\lg n)$ time respectively if the values of all elements are bounded by n. Finally, Nekrich and Vitter [22] presented an $O(n)$-words static data structure that answers queries in $O(k+1)$ time; their result is valid even in the case when point are not in the rank space. They also presented a dynamic version of their structure that uses the same space and achieves the same query time while handling updates in $O(\lg^\varepsilon n)$ time.

One-dimensional color counting in the rank space was studied by Gagie et al. [11]. They gave a data structure that answers queries in $O(\lg^{1+\varepsilon} n)$ time for any constant $\varepsilon > 0$ and uses $nH_0(S) + O(n) + o(nH_0(S))$ bits. Nekrich [21] described a data structure that uses $O(n\lg n)$ bits and answers color counting queries in $O(\lg k/\lg\lg n)$ time, where k is the number of colors. A lower bound that follows from the predecessor problem [1, 3] holds for exact one-dimensional color counting, and does not permit constant query time for a data structure with space bounded by a polynomial function of n. We circumvent this lower bound by focusing on approximate color counting. If we combine a reduction of one-dimensional color counting to point counting in 2D with the result of Chan and Wilkinson [7], we obtain a data structure that uses $O(n\lg n)$ bits and answer approximate color counting queries in $O(\lg^\varepsilon n)$ time. The data structure of Nekrich [7] also uses $O(n\lg n)$ bits but answers approximate color counting queries in $O(1)$ time. In both [21] and [7] it is assumed that points are in the rank space. In the general case, Saladi [24] presented a data structure that uses $O(n)$ words and answers queries in $O(\lg\lg U)$ time.

Our Results. We focus on studying one-dimensional color reporting and counting in the succinct scenario. In Section 2 we solve an open problem from [24] by presenting a data structure that answers approximate color counting queries in optimal $O(1)$ time. Our data structure uses $\mathcal{B}(n,m) + O(n) + o(\mathcal{B}(n,m))$ bits, where $\mathcal{B}(n,m) = n\lg(m/n)$ is the minimum number of bits required to store a set of size n from a universe of m elements. Thus, we demonstrate that is not necessary to store the colors of points in order to answer approximate color counting queries. If points are in the rank space, our data structure needs only $O(n)$ bits and does not require access to the original data set. That is, similar to data structures for answering range minimum queries [9] that can answer queries without storing the original data set, we can construct a data structure for a colored set of points S and discard the set S.

Using our data structure, we are still able to obtain a constant factor approximation on the number of colors in $S \cap [a, b]$ for an arbitrary query interval $[a, b]$.

Then we turn to the problem of reporting colors using succinct space. We describe a data structure that answers color reporting queries in $O(k + 1)$ time while using $\mathcal{B}(n, m) + nH_d(S) + o(\mathcal{B}(n, m) + n \lg \sigma)$ bits in Section 4. This result is a succinct counterpart of the data structure from [22] that also achieves optimal query time but uses $O(n \lg n)$ bits.

Finally we consider dynamic succinct color reporting in the rank space. We present a succinct data structure that answers color reporting queries in optimal $O(k + 1)$ time and updates in $O(\lg n)$ time while using $nH_d(S) + o(n \lg \sigma)$ bits. Our data structure supports an update operation that changes the color of a point in $O(\lg n)$ time.

Applications. Color reporting and counting queries are related to problems that arise in string processing and databases. Color searching queries are helpful when we are interested in (the number of) distinct object categories in a query range or look for distinct documents that contain a query substring. One prominent example is the document counting queries on a collection of documents. We keep documents (strings) $d_1, \ldots, d_D$ in a data structure so that for any query string P the number of documents that contain P can be calculated. This problem can be solved by answering color counting queries on the so called document array; see [20, 12] for a detailed description. The document array, however, needs $O(n \lg D)$ bits of space in the worst case. If the number of documents is large and the alphabet size is small, the space usage of the document array can be significantly larger than the space needed to store the document collection. Using the result of Theorem 4, we can answer approximate document counting queries using $O(n)$ additional bits.

In this paper we assume that the reader is familiar with basic concepts of succinct data structures and range reporting.

2 Approximate Color Range Counting

In this section we present a data structure that uses $\mathcal{B}(n, m) + O(n) + o(\mathcal{B}(n, m))$ bits of space and answers approximate color counting queries in constant time. A color range counting query for an interval returns the number of distinct colors contained within the interval. For any constant $\varepsilon > 0$, our color range counting data structure returns in constant time an approximate answer of at most $(1 + \varepsilon)$ of the correct answer.

2.1 Approximate Color Range Counting in Rank Space

We begin by describing a data structure for the problem in the special case when the input points are in the rank space. The input consists of a sequence $S = s_1, \ldots, s_n$ of n colors. A query is a range $[a, b]$ where $a, b \in [n]$, and the answer is a $(1 + \varepsilon)$-approximation of the number of distinct colors found in $s_a, \ldots, s_b$.

2.1.1 Space Inefficient Solution

First we describe a space inefficient solution that requires $O(n \lg^3 n)$ bits of space and answers one-dimensional approximate color counting queries in constant time.

Consider the balanced binary tree $\mathcal{T}$, where every leaf of $\mathcal{T}$ corresponds to an element of S, and every internal node has two children. Given a node $u \in \mathcal{T}$, $u_l(u_r)$ denotes the left(right) child of u, $S(u)$ denotes the set of all elements stored in the leaf descendants of u, and $a_u(b_u)$ denotes the rightmost(leftmost) element in $S(u_l)(S(u_r))$.

Let $\delta = 1 + \varepsilon$. For each node $u \in \mathcal{T}$ we store the unique values $l_1, \ldots, l_{\log_\delta n}$ in a fusion tree [10], where l_i $(1 \leq i \leq \log_\delta n)$ is the maximum value satisfying the condition that $s_{l_i}, \ldots, s_{a_u}$ contains δ^i unique colors. Also, for each node $u \in \mathcal{T}$ and each i $(1 \leq i \leq \log_\delta n)$ we store the unique values $r_{i1}, \ldots, r_{i\log_\delta n}$ in a fusion tree [10], where r_{ij} $(1 \leq j \leq \log_\delta n)$ is the minimum value satisfying the condition that $s_{b_u}, \ldots, s_{r_j}$ contains δ^j unique colors that are not present in $s_{l_i}, \ldots, s_{a_u}$.

Query. Given a query $[a, b]$ we find the lowest common ancestor u of a and b in $\mathcal{T}$. We query the fusion tree stored on $l_1, \ldots, l_{\log_\delta n}$ to find the predecessor l_i of a, then we query the fusion tree stored on $r_{i1}, \ldots, r_{i\log_\delta n}$ and find the successor r_{ij} of b. Finally we return $\delta^i + \delta^j$ as an estimate for the number of distinct colors in $[a, b]$.

▶ **Lemma 1.** *The algorithm described above returns a $(1 + \varepsilon)$-approximation of the number of distinct colors in $s_a, \ldots, s_b$.*

Proof. Denote by x the number of distinct colors in $s_a, \ldots, s_{a_u}$ and y the number of distinct colors in $s_{b_u}, \ldots, s_b$ that are not found in $s_a, \ldots, s_{a_u}$. Let y' denote the number of colors in $s_{b_u}, \ldots, s_b$ that do not occur in $l_i, \ldots, s_{a_u}$. By the definition of l_i and r_{ij}, $x \leq \delta^i \leq \delta \cdot x$ and $y' \leq \delta^j \leq \delta \cdot y'$. Since $y' \leq y$, $\delta^j \leq \delta \cdot y$. Hence $\delta^i + \delta^j \leq \delta(x + y)$. There are at most $\delta^i - x$ colors that occur in $l_i, \ldots, s_{a_u}$, but do not occur in $s_a, \ldots, s_{a_u}$. Hence $y - (\delta^i - x) \leq y'$ and $y - (\delta^i - x) \leq \delta^j$. If we add δ^i to both parts of the latter inequality, we obtain $y + x \leq \delta^j + \delta^i$. Summing up

$$x + y \leq \delta^i + \delta^j \leq \delta(x + y)$$

which completes the proof. ◀

▶ **Theorem 2.** *There exists an $O(n \lg^3 n)$-bit data structure that supports one-dimensional $(1 + \varepsilon)$-approximate color range counting queries in constant time when the input points are in the rank space.*

2.1.2 Lower Bound

Next, we show using a simple proof that $\Omega(n)$ bits are required for any data structure that answers one-dimensional $(1 + \varepsilon)$-approximate color range counting queries in the rank space.

We assume without loss of generality that $\sigma > \lfloor 1 + \varepsilon \rfloor$, otherwise no data structure is needed since returning σ for any query would be a correct $(1 + \varepsilon)$-approximation of the exact answer. Moreover, denote by $c_1, c_2, \ldots, c_k$ the first $k = \lfloor 1 + \varepsilon \rfloor + 1$ colors. Divide a sequence S of size n to n/k blocks each of size k. We say that S satisfies property $(*)$ if for each block b in S one of the following two conditions hold:

- either b consists of the color c_1 repeated k times,
- or $b = c_1, c_2, \ldots, c_k$.

Clearly, the number of sequences that satisfy $(*)$ is $2^{(n/k)}$ since there exist n/k blocks in a sequence of size n and each block can have one of two different values. Moreover for any two distinct sequences S_1 and S_2 satisfying $*$ differing at block b, there exist at least one $(1 + \varepsilon)$-approximate range counting query, namely the query that asks for the number of different colors in b, that will return different values. Thus, the information theoretic lower bound for storing a one-dimensional $(1 + \varepsilon)$-approximate range counting data structure is $\Omega(\lg 2^{(n/k)}) = \Omega(n/k) = \Omega(n/\epsilon)$ bits.

▶ **Theorem 3.** *Any one-dimensional $(1 + \varepsilon)$-approximate range counting data structure requires $\Omega(n/\epsilon)$ bits.*

2.1.3 Compact Data Structure

In this subsection we show how to make the data structure of Theorem 2 compact by bootstrapping. Let $\delta = 1 + \varepsilon$. We define the functions $f(n) = \lg^4 n$, $f^{(h)}(n) = f^{(h-1)}(f(n))$. The function $f^*(n)$ is defined as $f^*(n) = 1 + f^*(f(n))$ for $n > 2^{16}$ and $f^*(n) = 1$ otherwise. We start by modifying the tree $\mathcal{T}$ so that each leaf of $\mathcal{T}$ corresponds to a block of $f(n)$ consecutive elements of S (instead of a single element of S). Then, we define the family of trees $\mathcal{T}_{ij}$ where $1 \leq i \leq f^*(n)$ and $1 \leq j \leq n/f^{(i)}(n)$ as follows. Tree $\mathcal{T}_{ij}$ spans the i^{th} block of S of size $f^{(i)}(n)$ (i.e. $s_{((i-1)f^{(i)}(n)+1)}, \ldots, s_{(if^{(i)}(n))}$) and each leaf of $\mathcal{T}_{ij}$ correspond to a block of $f^{(i+1)}(n)$ consecutive elements. For each node $u \in \mathcal{T}_{ij}$ we store in separate fusion trees the sets of values: $\{l_p | 1 \leq p \leq \log_\delta f^{(i)}(n)\}$, and for each $1 \leq p \leq \log_\delta f^i(n)$ the set $\{r_{pq}|, 1 \leq q \leq \log_\delta f^{(i)}(n)\}$ as defined in Section 2.1.1. Finally, for every two indices a and b satisfying $1 \leq a \leq b \leq f(n)$ we store in a table B the index i such that a and b are in the same block of size $f^i(n)$ but in different blocks of size $f^{i+1}(n)$. In other words, i must satisfy the following conditions $\lfloor a/f^{(i)}(n) \rfloor = \lfloor b/f^{(i)}(n) \rfloor$ and $\lfloor a/f^{(i+1)}(n) \rfloor \neq \lfloor b/f^{(i+1)}(n) \rfloor$

Space Analysis. The number of nodes in $\mathcal{T}$ is reduced to $n/f(n)$ and the space used by $\mathcal{T}$ and fusion trees stored in its nodes is $O(n/\lg n)$ bits. The number of nodes in $\mathcal{T}_{ij}$ is $f^{(i)}(n)/f^{(i+1)}(n)$ and the space used by $\mathcal{T}_{ij}$ and fusion trees stored in its nodes is $O(f^{(i)}(n)/\lg(f^{(i)}(n)))$ bits. Thus, the total space used by all such trees is:

$$\begin{aligned}
\sum_{i=1}^{f^*(n)} \left(\sum_{j=1}^{n/f^{(i)}(n)} O\Big(f^{(i)}(n)/\lg(f^{(i)}(n))\Big) \right) &= \sum_{i=1}^{f^*(n)} \Big(n/f^{(i)}(n) \cdot O\Big(f^{(i)}(n)/\lg(f^{(i)}(n))\Big) \Big) \\
&= \sum_{i=1}^{f^*(n)} O\Big(n/\lg(f^{(i)}(n))\Big) \\
&= n \sum_{i=1}^{f^*(n)} O\Big(1/\lg(f^{(i)}(n))\Big) \\
&= O(n)
\end{aligned}$$

Finally, the table B uses $o(n)$ bits. Thus, the total space used is $O(n)$ bits.

Query. Given a query $[a, b]$, if a and b are in two different blocks of size $f(n)$, we can answer queries using $\mathcal{T}$ in the same way as described in Subsection 2.1.1. Otherwise, we query B on values $(a \mod f(n))$ and $(b \mod f(n))$ to find the index i satisfying the condition that a and b are in the same block of size $f^{(i)}(n)$ but in different blocks of size $f^{(i+1)}(n)$. Finally, we query $\mathcal{T}_{i\lfloor a/f^{(i)}n \rfloor}$ as we query $\mathcal{T}$.

▶ **Theorem 4.** *There exists a compact $O(n)$-bit data structure that supports one-dimensional $(1+\varepsilon)$-approximate color range counting queries in constant time when the input points are in the rank space.*

3 General Approximate Range Counting

In this section, we present a data structure that uses $\mathcal{B}(n, m) + O(n) + o(\mathcal{B}(n, m))$ bits of space and answers $(1 + \varepsilon)$-approximate color counting queries in constant time.

Let $\delta = 1 + \varepsilon$ and let $x_1, \ldots, x_n$ be the coordinates of the n given colored points $\mathcal{P}$ in sorted order. Denote by $\mathcal{P}_{\lceil \lg^3 n \rceil}$ the set of points whose x-coordinate rank is a multiple of

$\lceil \lg^3 n \rceil$. For each point $p \in \mathcal{P}$ denote by $\mathrm{L}(p)$ the set of points to the left of p, and by $\mathrm{R}(p)$ the set of points to the right of p.

For each point $p \in \mathcal{P}_{\lceil \lg^3 n \rceil}$ we store in a fusion tree [10] the unique values $l_1, \ldots, l_{\log_\delta n}$ where l_i ($i \in [\log_\delta n]$) is the maximum value satisfying the condition that $s_{l_i}, \ldots, s_p$ contains δ^i unique colors. Also, for each point $p \in \mathcal{P}_{\lceil \lg^3 n \rceil}$ and each $i \in [\log_\delta n]$ we store in a fusion tree [10] the unique values $r_{i1}, \ldots, r_{i \log_\delta n}$ where r_{ij} ($j \in [\log_\delta n]$) is the minimum value satisfying the condition that $s_{p+1}, \ldots, s_{r_j}$ contains δ^j unique colors not present in $s_{l_i}, \ldots, s_p$. We also store a succinct point reporting structure [13] on $\mathcal{P}_{\lceil \lg^3 n \rceil}$.

Next, we divide $x_1, \ldots, x_n$ into $n/\lceil \lg^3 n \rceil$ blocks each of size $\lceil \lg^3 n \rceil$, except for the last one. Using $O(n \lg^{4/5} m)$ bits [23] we store predecessor and successor data structures for each block independently. Since the size of each block is at most $\lceil \lg^3 n \rceil$, answering predecessor and successor queries within a block takes constant time. Finally, we store in $O(n)$ bits the compact data structure from Theorem 4 for answering queries in the rank space.

Query. Given a query $[a, b]$ we check if a point $p \in \mathcal{P}_{\lceil \lg^3 n \rceil}$ is in $[a, b]$. If so, we query the fusion tree stored on $l_1, \ldots, l_{\log_{1+\varepsilon} n}$ to find l_i the predecessor of a, then we query the fusion tree stored on $r_{i1}, \ldots, r_{i \log_{1+\varepsilon} n}$ to find r_{ij} the successor of b, afterwards we return $(1+\varepsilon)^i + (1+\varepsilon)^j$.

If such a point p does not exist, then both a and b are in one of the blocks whose size is $\lceil \lg^3 n \rceil$. Using the reporting data structure stored on $\mathcal{P}$ we get the rank of an arbitrary point in $[a, b]$ then determine which block does a and b belong to. Afterwards, using the predecessor and successor structures, we determine the rank of a and b. Since the query is now reduced to the rank space, we can answer it in constant time.

▶ **Theorem 5.** *There exists an $(\mathcal{B}(n, m) + O(n) + O(n \lg^{4/5} m))$-bit data structure that supports one-dimensional $(1+\varepsilon)$-approximate color range counting queries in constant time.*

Next, we describe how to reduce the space of the predecessor and successor data structures. We use a well known trick and split the universe $[m]$ into n subranges $r_1, \ldots, r_n$ each of size m/n. We also use succinct rank and select data structures that store a bit vector of size n using $n + o(n)$ bits and answers rank and select queries in constant time [19]. For each non-empty subrange r_i we store a predecessor and successor structure for every block of $\lg^2 n$ consecutive elements and a point reporting structure P_i on all the points within r_i. These structures are stored consecutively in an array A. To locate the data structures for any range r_i within A, we count the number of points in the ranges r_j for $j < i$ then scale that number. For that purpose, we construct a bit vector B of size $2n$ bits, with rank and select queries, that stores a zero for each range r_i followed by n_i ones, where n_i is the number of points in the range r_i. To count the number of points preceding r_i, we use a select query to get the position k of the i^{th} zero in B, then with a rank query we count the number of ones before position k.

Given a non-empty query range $[a, b]$ such that there exist at most $\lg^3 n$ points between a and b, a belongs to r_i where $i = \lfloor a/(m/n) \rfloor$ and b belongs to r_j where $j = \lfloor b/(m/n) \rfloor$, we find the rank of a in the following manner. First, we map a to $a' = a - im/n$ and b to $b' = b - jm/n$. If the range $[a', m/n]$ is empty in P_i, we use rank and select queries to get s the number of ones before the $(i+1)^{\text{th}}$ zero in B, the rank of a will be $s + 1$. Otherwise, we find a point p in P_i within the range $[a', m/n]$ if i and j are different or within the range $[a', b']$ if i and j are the same. If p's rank within r_i is k, we query the $\lfloor k/\lg^3 n \rfloor$ successor data structure to find the rank of a' in r_i. Then, we add the number of points occurring in each range r_l where $l < i$ to this rank to get the rank of a. We obtain the rank of b in a similar manner.

The extra space used is $o(\mathcal{B}(n,m))$ bits for the point reporting structures stored on the ranges $r_1,\ldots,r_n$, $O(n\lg^{4/5}(m/n)) = o(\mathcal{B}(n,m))$ bits for the predecessor and successor data structures, and $O(n)$ bits for the bit vector B.

▶ **Theorem 6.** *There exists an $(\mathcal{B}(n,m)+O(n)+o(\mathcal{B}(n,m))$-bit data structure that supports one-dimensional $(1+\varepsilon)$-approximate color range counting queries in constant time.*

4 1D Color Range Reporting

Using similar techniques to those used in the previous section, we present in this section a succinct data structure that uses $\mathcal{B}(n,m)+nH_d(S)+o(\mathcal{B}(n,m)+n\lg\sigma)$ bits of space and answers color reporting queries in optimal $O(k+1)$ time.

If the input points are in the rank space (i.e. the x-coordinates of the input points are 1,...,n and the input consists of a sequence $S=s_1,\ldots,s_n$ of n colors, a query is a range $[a,b]$ where $a,b\in[n]$, and the answer is the distinct colors found in $s_a,\ldots,s_b$), one-dimensional color range reporting can be solved in $O(k+1)$ time using $nH_d(S)+o(n)\lg\sigma+O(n\lg\lg\sigma)$ bits [2, 4, 6].

This solution can be extended to general one-dimensional range reporting by storing the x-coordinates of the points in sorted order in an indexable dictionary that supports select queries in constant time using $\mathcal{B}(n,m)+o(\mathcal{B}(n,m))$ bits [25] in addition to the data structure described in [2, 4, 6]. We can find the predecessor or successor of any x-coordinate in $O(\lg n)$ time by answering $O(\lg n)$ select queries. Hence, we can reduce any query $[a,b]$ to the rank space in $O(\lg n)$ additional time.

▶ **Theorem 7.** *There exists an $(\mathcal{B}(n,m)+nH_d(S)+o(\mathcal{B}(n,m)+n\lg\sigma))$-space data structure that supports one-dimensional color range reporting queries in $O(\lg n+k)$time.*

4.1 Improved Data Structure

Next, we show how to improve the query time obtained from Theorem 7 to $O(k+1)$, while using the same amount of space.

Let $x_1,\ldots,x_n$ be the coordinates in sorted order of the n given colored points $\mathcal{P}$. We denote by $\mathcal{P}_{\lceil\lg^2 n\rceil}$ the set of points whose x-coordinate rank is a multiple of $\lceil\lg^2 n\rceil$. For each point $p\in\mathcal{P}$ we denote by $\mathrm{L}(p)$ the set of points to the left of p, and by $\mathrm{R}(p)$ the set of points to the right of p. For every color z the set $\mathrm{Min}(p)$ contains the minimal element $e\in\mathrm{L}(p)$ of color z, and the set $\mathrm{Max}(p)$ contains the maximal element $e\in\mathrm{R}(p)$ of color z.

Data Structure. For each point $p\in\mathcal{P}_{\lceil\lg^2 n\rceil}$, we store the smallest $\lceil\lg n\rceil$ elements of $\mathrm{Min}(p)$ and the largest $\lceil\lg n\rceil$ elements of $\mathrm{Max}(p)$. We also store two succinct one-dimensional point reporting data structures [13], one on every point in $\mathcal{P}$, and the other on every point in $\mathcal{P}_{\lceil\lg^2 n\rceil}$. Next, we store a data structure similar to the one used in subsection 3 that can find in constant time the ranks of a query $[a,b]$ if $[a,b]$ is not empty, and a and b belong to the same block of size $\lg^2 n$, Finally, we store the data structure from Theorem 7.

Answering Queries. We report all colors in a query range $[a,b]$ as follows. Using the reporting data structure stored on $\mathcal{P}_{\lceil\lg^2 n\rceil}$, we search for some $p\in\mathcal{P}_{\lceil\lg^2 n\rceil}\cap[a,b]$.

If such a point p exist, we traverse the list $\mathrm{L}(p)$ until an element $p'>b$ is found or the end of $\mathrm{L}(p)$ is reached. We also traverse the list $\mathrm{R}(p)$ until an element $p'<a$ is found or the end of $\mathrm{R}(p)$ is reached. If we reach neither the end of $\mathrm{L}(p)$ nor the end of $\mathrm{R}(p)$, then

all distinct colors in $[a, b]$ are reported. Otherwise, the range $[a, b]$ contains more than $\lg n$ distinct colors. In that case we use the data structure from Theorem 7.

If a and b belong to a continuous block of $\lg^2 n$ points, we find their ranks in a similar manner to subsection 3, then solve the problem in the rank space as described in the previous subsection.

▶ **Theorem 8.** *There exists a* $(\mathcal{B}(n, m) + nH_d(S) + O(n) + o(\mathcal{B}(n, m) + n \lg \sigma))$*-bit data structure that supports one-dimensional color range reporting queries in* $O(k + 1)$ *time.*

Note that $n = o(n \lg \sigma)$ as long as σ is not a constant. If σ is a constant, we solve the problem using a different approach. We store a separate succinct range emptiness data structure [13] for every subset of points with a given color. To answer a query $[a, b]$, for each color c we query the range emptiness data structure associated with c to check if a point with color c occurs in the range $[a, b]$, if so we report c. The query runtime is a constant since the number of colors is constant and range emptiness queries take constant time. Hence, we obtain the following theorem.

▶ **Theorem 9.** *There exists an* $(\mathcal{B}(n, m) + nH_d(S) + o(\mathcal{B}(n, m) + n \lg \sigma))$*-space data structure that supports one-dimensional color range reporting queries in* $O(k + 1)$ *time.*

5 Dynamic Color Reporting in Rank Space

Finally, we describe a succinct data structure that uses $nH_d(S) + o(n \lg \sigma)$ bits of space and answers color reporting queries in optimal $O(k + 1)$ time when the input points are in the rank space, while supporting the following update operation in $O(\lg n)$ time: given an index i and a color c, set the color of the i^{th} element to c.

▶ **Theorem 10.** *There exists an* $(nH_d(S) + o(n \lg \sigma) + O(n))$*-bit data structure that supports one-dimensional color range reporting queries in* $O(k + 1)$ *time and updates in* $O(\lg n)$ *time when points are in the rank space.*

Proof. Let the input sequence be $S = s_1, \ldots, s_n$, and $\mathcal{T}$ be the complete balanced binary tree where every leaf of $\mathcal{T}$ corresponds to an element of S and every internal node has two children. For any node $u \in \mathcal{T}$, $S(u)$ denotes the set of all elements stored in the leaf descendants of u. For $i \in \{1, \ldots, n\}$ denote by $l_i(r_i)$ the height of the highest ancestor u of the node corresponding to i such that i is the leftmost(rightmost) element in $S(u)$ with color s_i.

We store S in a dynamic data structure using $nH_d(S) + o(n \lg \sigma)$ bits that supports access in $O(1)$ time and Update, Rank, and Select in $O(\lg n / \lg \lg n)$ time [14]. We divide S into blocks of $\lg n$ elements each, then we subdivide each block to subblocks of size $\lg \lg n$ elements. For each subblock b_{ij} $(0 \leq i < n / \lg n$ and $0 \leq j < \lg n / \lg \lg n)$ in block b_i we store:

- The maximum value m^l_{ij} of the sequence $l_{i \lg n + j \lg \lg n}, \ldots, l_{i \lg n + (j+1) \lg \lg n}$ and a succinct range maximum data structure [9] T^l_{ij} to answer range maximum queries on it.
- The maximum value m^r_{ij} of the sequence $r_{i \lg n + j \lg \lg n}, \ldots, r_{i \lg n + (j+1) \lg \lg n}$ and a succinct range maximum data structure [9] T^r_{ij} to answer range maximum queries on it.

The space used is $O(\lg \lg n)$ bits per subblock, which sums to $O(n)$ bits. For each block b_i we store:

- The sequence $m^l_{i0}, \ldots, m^l_{i \lg \lg n}$, its maximum value m^l_i, and a succinct range maximum data structure [9] T^l_i to answer range maximum queries on it.

- The sequence $m^r_{i0}, \ldots, m^r_{i \lg\lg n}$, its maximum value m^r_i, and a succinct range maximum data structure [9] T^r_i to answer range maximum queries on it.

The space used is $O(\lg n/\lg\lg n)$ bits per block, which sums to $O(n/\lg\lg n)$ bits. Finally, using Lemma 1 from a result by Nekrich et al. [22] we store using $O(n)$ bits two-dimensional point reporting structures T^l and T^r containing the set of points (i, m^l_i) and (i, m^r_i) where $1 \leq i \leq n/\lg n$. These structures support queries in $O(k+1)$ time and updates in $O(\lg^\varepsilon n)$ time.

Answering Queries: Given a query $[a, b]$, we find the lowest common ancestor u of a and b. Let $u_l(u_r)$ be the left(right) child of u, c be the rightmost child of u_l, and let h denote the height of u_l and u_r.

To get all distinct colors in $[a, c] = [a, b] \cap S(u_l)$, it is sufficient to report all colors s_i in that range with $r_i \geq h$. We maintain the invariant that each color is reported on its right most occurrence.

If $[a, c]$ was contained in a single subblock b_{ij}, we query T^r_{ij} for all the distinct colors as follows. We get the largest element r_d in $r_a, \ldots, r_c$, if s_d was previously reported we return, otherwise we report s_d and recurse on the interval $[d, c]$ followed by $[a, d]$. Note that it is important to recurse on $[d, c]$ before $[a, d]$ to maintain the invariant mentioned above, which guarantees that $r_d = \min(r_a, \ldots, r_c)$ will be smaller than h if the color s_d was previously reported.

Otherwise, if $[a, c]$ spans several subblocks but is contained in a single block b_i we proceed as follows. We first query the rightmost subblock partially spanned by $[a, c]$. Then, we query T^r_i to get all the subblocks b_{ij} spanned by $[a, c]$ satisfying the condition that $m^r_{ij} \geq h$ in order from right to left. We query each one of them in that order, then we query the leftmost subblock that is partially spanned by $[a, c]$.

Finally, if $[a, c]$ spans several blocks we first query the rightmost block partially spanned by $[a, c]$. Then, we query T^r to get all the blocks i spanned by $[a, c]$ satisfying the condition that $m^r_i \geq h$ in order from right to left. We query each one of them in that order, then we query the leftmost block that is partially spanned by $[a, c]$.

Similarly, to report all the distinct colors in $[c+1, b] = [a, b] \cap S(u_r)$ it is sufficient to report all colors s_i in that range with $l_i \geq h$. We do this in a similar way to the method used to query $[a, c]$, while maintaining the invariant that each color is reported on its left most occurrence.

Updating the Sequence: If the color of position i was updated from c to c' the following values could get modified: r_i, r_a where a is the first index before i with color c, r_b where b is the first index after i with color c', l_i, l_d where d is the first index after i with color c, and l_e where e is the first index before i with color c'.

We can find the value r_i of any index i in $O(\lg n/\lg\lg n)$ time by using Rank and Select queries to get the first index j before i with the same color as index i, then computing the lowest common ancestor of i and j. Similarly, to get the value l_i, we use Rank and Select queries to get the first index j after i with the same color as index i, then we compute the lowest common ancestor of i and j.

Since we don't store the values $r_1, \ldots, r_n$ and $l_1, \ldots, l_n$ explicitly, once one of them changes (say r_a where a is in subblock b_{ij}) we recompute all values r_j where $j \in b_{ij}$ and reconstruct T^r_{ij}. Recomputing all values r_j where $j \in b_{ij}$ takes $O(\lg\lg n \cdot \lg n/\lg\lg n) = O(\lg n)$ time and reconstructing T^r_{ij} takes $O(\lg\lg n)$ time. If m^r_{ij} changed, we rebuild T^r_i in $O(\lg n)$ time. Finally, if m_i changed we update its value in T^r in $O(\lg^\varepsilon n)$ time. Since only a constant number of values get updated, the runtime is $O(\lg n)$. ◀

If σ is a constant then the $O(n)$ additional bits stored by the data structure are no longer a lower order term, so we handle this case separately. We divide S into blocks of size $\lg n/2\lg\sigma$. We store a lookup table using $O(\sqrt{n}\lg^2 n)$ bits to answer color range queries over every possible block of this size. Also, we store the data structure from Theorem 10 on the sequence $S' = s'_1, \ldots, s'_{2\lg\sigma n/\lg n}$ with alphabet $\sigma' = 2^\sigma$, where s'_i denotes the subset of colors found on the i^{th} block of S. The total space used is $nH_d(S) + o(n\lg\sigma) + O(n/\lg n)$ bits. To answer a query $\mathcal{Q}$, we use the lookup table to get the colors in the (two) blocks which are not completely spanned by $\mathcal{Q}$, then we use the data structure from Theorem 10 to get the colors in the blocks that are fully spanned by $\mathcal{Q}$. Each color will be reported at most a constant number of times. The query time is $O(k+1) = O(1)$ and update time is $O(\lg n)$.

▶ **Theorem 11.** *There exists an $(nH_d(S) + o(n\lg\sigma))$-bit data structure that supports one-dimensional color range reporting queries in $O(k+1)$ time and updates in $O(\lg n)$ time when points are in the rank space.*

Acknowledgment. The first author gratefully acknowledges the discussions he had with Rahul Saladi on the problem.

References

1 Miklós Ajtai. A lower bound for finding predecessors in Yao's cell probe model. *Combinatorica*, 8(3):235–247, 1988.

2 Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

3 Paul Beame and Faith E Fich. Optimal bounds for the predecessor problem. In *Proceedings of the 31st annual ACM symposium on Theory of computing*, pages 295–304. ACM, 1999.

4 Philip Bille, Gad M Landau, Rajeev Raman, Kunihiko Sadakane, Srinivasa Rao Satti, and Oren Weimann. Random access to grammar-compressed strings. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 373–389. Society for Industrial and Applied Mathematics, 2011.

5 Prosenjit Bose, Eric Y. Chen, Meng He, Anil Maheshwari, and Pat Morin. Succinct geometric indexes supporting point location queries. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009*, pages 635–644, 2009.

6 Panayiotis Bozanis, Nectarios Kitsios, Christos Makris, and Athanasios Tsakalidis. New upper bounds for generalized intersection searching problems. In *Proceedings of the 22nd International Colloquium on Automata, Languages, and Programming*, pages 464–474. Springer, 1995.

7 Timothy M Chan and Bryan T Wilkinson. Adaptive and approximate orthogonal range counting. *ACM Transactions on Algorithms (TALG)*, 12(4):45, 2016.

8 Arash Farzan, J. Ian Munro, and Rajeev Raman. Succinct indices for range queries with applications to orthogonal range maxima. In *Proceedings of the 39th International Colloquium on Automata, Languages, and Programming, Part I*, pages 327–338, 2012.

9 Johannes Fischer. Optimal succinctness for range minimum queries. In *Proceedings of the 9th Latin American Symposium on Theoretical Informatics*, pages 158–169. Springer, 2010.

10 Michael L Fredman and Dan E Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47(3):424–436, 1993.

11 Travis Gagie and Juha Kärkkäinen. Counting colours in compressed strings. In *Proceedings of the 22nd Annual Symposium on Combinatorial Pattern Matching, CPM 2011*, pages 197–207, 2011.

12 Travis Gagie, Juha Kärkkäinen, Gonzalo Navarro, and Simon J Puglisi. Colored range queries and document retrieval. *Theoretical Computer Science*, 483:36–50, 2013.

13 Mayank Goswami, Allan Grønlund Jørgensen, Kasper Green Larsen, and Rasmus Pagh. Approximate range emptiness in constant time and optimal space. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 769–775, 2015.

14 Roberto Grossi, Rajeev Raman, Srinivasa Rao Satti, and Rossano Venturini. Dynamic compressed strings with random access. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming, Part I*, pages 504–515, 2013.

15 Meng He. Succinct and implicit data structures for computational geometry. In *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, pages 216–235, 2013.

16 J Ian Munro and S Srinivasa Rao. Succinct representation of data structures. In *Handbook of Data Structures and Applications*, chapter 37. Chapman and Hall/CRC, 2004.

17 Ravi Janardan and Mario Lopez. Generalized intersection searching problems. *International Journal of Computational Geometry & Applications*, 3(01):39–69, 1993.

18 Christian W Mortensen. Generalized static orthogonal range searching in less space. Technical report, Citeseer, 2003.

19 J Ian Munro. Tables. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 37–42. Springer, 1996.

20 S Muthukrishnan. Efficient algorithms for document retrieval problems. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 657–666. Society for Industrial and Applied Mathematics, 2002.

21 Yakov Nekrich. Efficient range searching for categorical and plain data. *ACM Trans. Database Syst.*, 39(1):9:1–9:21, 2014.

22 Yakov Nekrich and Jeffrey Scott Vitter. Optimal color range reporting in one dimension. In *Proceedings of the 21st European Symposium on Algorithms*, pages 743–754. Springer, 2013.

23 Mihai Patrascu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 232–240, 2006.

24 Saladi Rahul. Approximate range counting revisited. *arXiv preprint arXiv:1512.01713*, 2015.

25 Rajeev Raman, Venkatesh Raman, and S Srinivasa Rao. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 233–242, 2002.

26 Michiel Smid and Prosenjit Gupta. Further results on generalized intersection searching problems: counting, reporting, and dynamization. Technical report, 1992.

27 Peter van Emde Boas. Preserving order in a forest in less than logarithmic time. In *Proccedings of the 16th Annual Symposium on Foundations of Computer Science*, pages 75–84. IEEE, 1975.