# Indexed Geometric Jumbled Pattern Matching

Stephane Durocher[1,⋆], Robert Fraser[1], Travis Gagie[2,⋆⋆], Debajyoti Mondal[1], Matthew Skala[1], and Sharma V. Thankachan[3]

[1] Department of Computer Science, University of Manitoba, Canada
{durocher, fraser, jyoti, mskala}@cs.umanitoba.ca
[2] Department of Computer Science, University of Helsinki, Finland
gagie@cs.helsinki.fi
[3] Cheriton School of Computer Science, University of Waterloo, Canada
thanks@uwaterloo.ca

**Abstract.** We consider how to preprocess $n$ colored points in the plane such that later, given a multiset of colors, we can quickly find an axis-aligned rectangle containing a subset of the points with exactly those colors, if one exists. We first give an index that uses $o(n^4)$ space and $o(n)$ query time when there are $\mathcal{O}(1)$ distinct colors. We then restrict our attention to the case in which there are only two distinct colors. We give an index that uses $\mathcal{O}(n)$ bits and $\mathcal{O}(1)$ query time to detect whether there exists a matching rectangle. Finally, we give a $\mathcal{O}(n)$-space index that returns a matching rectangle, if one exists, in $\mathcal{O}(\lg^2 n / \lg\lg n)$ time.

## 1 Introduction

Over the past ten years, researchers have studied online jumbled pattern matching for strings, graphs and, most recently, point sets. Butman et al. [6] showed how, given a string and multiset of characters, in linear time we can find all the substrings consisting of a permutation of those characters. The multiset is usually represented as a Parikh vector of characters, i.e., a list of the distinct characters' frequencies. Lacroix et al. [16] showed how, given a node-colored tree and a Parikh vector of colors with a constant number of non-zero entries, in polynomial time we can find all the connected subgraphs whose nodes have exactly the prescribed colors. Fellows et al. [11] extended this result to graphs with bounded tree-width and, implicitly, to all graphs when the Parikh vector's L1 norm (i.e., the multiset's size) is at most logarithmic in the size of the graph [3]. Barba et al. [2] showed how, given a set of colored points in the plane and a Parikh vector of colors, in cubic time we can find all the minimal axis-aligned rectangles containing subsets of points with exactly the prescribed colors.

Over the past five years, researchers have also studied indexed jumbled pattern matching for strings and graphs. Cicalese et al. [8] showed how, given a binary string, in quadratic time we can build a linear-space index such that later, given frequencies of 0s and 1s, we can detect in constant time whether

⋆ Supported by the Natural Sciences and Engineering Research Council of Canada.
⋆⋆ Supported by the Academy of Finland.

there exists a matching substring. Subsequent authors [1, 5, 9, 14, 17] have reduced the construction time slightly for the general case, to $\mathcal{O}(n^2/\lg^2 n)$, and more significantly for special cases or relaxations. Gagie et al. [13] reduced the space from a linear number of words to a linear number of bits and extended the result to trees whose nodes each have one of two colors, still with $\mathcal{O}(n^2/\lg^2 n)$ construction time.

In this paper we study indexed jumbled pattern matching in point sets. In Section 2 we generalize Kociumaka et al.'s [15] recent proof that, given a string of length $n$ over a $\mathcal{O}(1)$-size alphabet, we can build a $o(n^2)$-space index such that later, given a Parikh vector of characters, in $o(n)$ time we can find a matching substring if one exists. We show that, given a set of $n$ points in the plane each of which is one of $\mathcal{O}(1)$ distinct colors, we can build a $o(n^4)$-space index such that later, given a Parikh vector of colors, in $o(n)$ time we can find one instance of a matching axis-aligned rectangle if one exists. Since these indexes provide an example match, we call them *witnessing* indexes. Notice that, when dealing with trees, graphs or point sets, the total number of matches can be superlinear, so returning only one instance seems reasonable. We will show in the full version of this paper how to derandomize our Las Vegas construction.

In Section 3 we generalize Cicalese et al.'s detection index for binary strings. We show how, given $n$ points in the plane in general position (i.e., with each x- and y-coordinate unique), each of which is one of two distinct colors, in $\mathcal{O}(n^3 \lg n)$ time we can build an index that occupies $\mathcal{O}(n)$ bits such that later, given a Parikh vector of those two colors, in $\mathcal{O}(1)$ time we can detect whether there exists a matching axis-aligned rectangle. As part of our construction, we use a new dynamic detection index for jumbled pattern matching in binary strings, with $\mathcal{O}(n \lg n)$ update time and $\mathcal{O}(1)$ query time. Although $\mathcal{O}(n \lg n)$-time updates may seem slow, we note that $o(n/\lg^2 n)$-time updates would imply a faster way to build static indexes for binary strings than that which is known. Due to space constraints, we leave the description of this dynamic index to full version of this paper, where we will also show that $\mathcal{O}(n)$-bit, $\mathcal{O}(1)$-time detection indexes exist not only for rectangles but for any shape that contains a point from which the entire interior is visible (i.e., star shapes).

In Section 4 we apply a recent technique by Cicalese et al. [10] to turn our detection index for bichromatic point sets into a witnessing index. We show how, given $n$ points in the plane in general position, each of which is one of two distinct colors, in $\mathcal{O}(n^3 \lg n)$ time we can build a $\mathcal{O}(n)$-space index such that later, given a Parikh vector of those two colors, in $\mathcal{O}(\lg^2 n/\lg\lg n)$ time we can return a matching axis-aligned rectangle, if one exists.

## 2 A Witnessing Index for $\mathcal{O}(1)$ Colors

It is possible to build a $\mathcal{O}(n^2)$-space index with $\mathcal{O}(1)$ query time for jumbled pattern matching in a string over a $\mathcal{O}(1)$-size alphabet, by building a perfect hash table of all its substrings' Parikh vectors; or to store nothing but the string itself and search it in $\mathcal{O}(n)$ time with Butman et al.'s algorithm for each query.

Nevertheless, Kociumaka et al.'s was the first (and, so far as we know, still the only) index for jumbled pattern matching in strings over ternary or larger alphabets, to use simultaneously $o(n^2)$ space and $o(n)$ query time. Specifically, for any alphabet size $\sigma = \mathcal{O}(1)$ and any positive $\epsilon < 1$, we can set their index to use $\mathcal{O}(n^{2-\epsilon})$ space and $\mathcal{O}(m^{(2\sigma-1)\epsilon})$ query time, where $m \leq n$ is the L1 norm of the Parikh vector in the query. Thus, choosing $\epsilon < 1/(2\sigma - 1)$ means we use $o(n^2)$ space and $o(m) = o(n)$ query time.

Similarly, it is possible to build a $\mathcal{O}(n^4)$-space index with $\mathcal{O}(1)$ query time for jumbled pattern matching in a point set, by building a perfect hash table of all the Parikh vectors of subsets that can be enclosed in axis-aligned rectangles; or to store nothing but the point set itself and search it in $\mathcal{O}(n^3)$ time with Barba et al.'s algorithm for each query. Therefore, an obvious starting point is to describe a $o(n^4)$-space index with $o(n^3)$ query time, analogous to Kociumaka et al.'s. This makes sense only for $\sigma \geq 4$ because, for $\sigma = 3$, we can store in $\mathcal{O}(n^3)$ space a perfect hash table of all Parikh vectors with L1 norm at most $n$.

In fact, for any positive $\epsilon < 1$, we can use $\mathcal{O}(n^{4-\epsilon})$ space and the same query time as Kociumaka et al.'s, $\mathcal{O}(m^{(2\sigma-1)\epsilon})$. Choosing again $\epsilon < 1/(2\sigma - 1)$, therefore, means we use $o(n^4)$ space and $o(n)$ query time.

**Theorem 1.** *Given a set of $n$ points in the plane each of which is one of a constant number $\sigma$ of distinct colors, we can store them in $\mathcal{O}(n^{4-\epsilon})$ space for any given positive $\epsilon < 1$ such that later, given a vector $\boldsymbol{C} = (c_1, \ldots, c_\sigma)$ with L1 norm $m$, in $\mathcal{O}(m^{(2\sigma-1)\epsilon})$ time we can return an axis-aligned rectangle containing exactly $c_i$ points of the $i$th color, for $1 \leq i \leq \sigma$, if such a rectangle exists.*

*Proof.* Without loss of generality, assume we are working in rank space (i.e., on an $n \times n$ grid with one point in each row and each column). For the moment, assume we know in advance values $b$ and $h$ such that $b \leq m \leq b + h$. For each axis-aligned rectangle containing between $b$ and $b + h$ points, we call the set of the lowest $b$ points the rectangle's *body* and we call the remaining points its *head*.

Consider all $\mathcal{O}(n^3)$ axis-aligned (not necessarily minimal) rectangles whose bottoms and tops hit points and that contain exactly $b$ points each. There are $\mathcal{O}(n^3)$ such rectangles because, once we have chosen the bottom point and left and right sides of such a rectangle, the fact it contains $b$ points determines which point its top must hit. We say a vector with L1 norm $b$ is *light* if there are between 1 and $t$ such rectangles that match that vector, where $t$ is a threshold we specify later; otherwise, we say the vector is *heavy*.

We store a perfect hash table of the light vectors and with each of those vectors, we store a list of the locations of the $\mathcal{O}(t)$ rectangles matching that vector. This takes a total of $\mathcal{O}(n^3)$ space regardless of $t$. We also store a $\mathcal{O}(n^2)$-space data structure that, given a 4-sided range, in $\mathcal{O}(1)$ time tells us how many points of each color lie in that range. We store another perfect hash table of all the distinct vectors for rectangles containing between $b$ and $b + h$ points whose bodies have heavy vectors; with each one, we store the location of one rectangle matching that distinct vector. Since there are $\mathcal{O}(n^3/t)$ heavy vectors and there are $\mathcal{O}(h^\sigma)$ possible distinct vectors for heads, this takes a total of $\mathcal{O}(n^3 h^\sigma / t)$ space.

Given $\boldsymbol{C}$, we consider all $\mathcal{O}(h^{\sigma-1})$ ways to choose two vectors $\boldsymbol{B}$ and $\boldsymbol{H}$ with non-negative entries such that $\boldsymbol{C} = \boldsymbol{B} + \boldsymbol{H}$, $\boldsymbol{B}$ has L1 norm $b$ and $\boldsymbol{H}$ has L1 norm at most $h$. For each choice of $\boldsymbol{B}$ and $\boldsymbol{H}$, we check whether $\boldsymbol{B}$ is light and, if so, we run through the list of the locations of the $\mathcal{O}(t)$ rectangles matching $\boldsymbol{B}$. For each such rectangle, we check whether the $|\boldsymbol{H}|$ points immediately above the rectangle match $\boldsymbol{H}$; if so, we return the location of the rectangle extended to include those $\boldsymbol{H}$ points, then stop. This takes $\mathcal{O}(t)$ time for each choice of $\boldsymbol{B}$ and $\boldsymbol{H}$. If $\boldsymbol{B}$ is not light then we search for $\boldsymbol{C}$ in our second perfect hash table, which takes $\mathcal{O}(1)$ time.

Overall, we use $\mathcal{O}(n^3 + n^3 h^\sigma / t)$ space and $\mathcal{O}(h^{\sigma-1} t)$ query time. Setting $t = h^\sigma$, our space becomes $\mathcal{O}(n^3)$ and our query time becomes $\mathcal{O}(h^{2\sigma-1})$. For any upper bound $M$ on $m$, if we repeat this construction for $b = 0, M^\epsilon, 2M^\epsilon, \ldots, M$ and $h = M^\epsilon$, then we use $\mathcal{O}(n^3 M^{1-\epsilon})$ space and we can answer any query with $m \leq M$ in $\mathcal{O}(M^{(2\sigma-1)\epsilon})$ time. Storing data structures for $M = 1, 2, 4, \ldots, n$ takes a total of $\mathcal{O}(n^{4-\epsilon})$ space and lets us answer any query in $\mathcal{O}(m^{(2\sigma-1)\epsilon})$ time. $\square$

## 3 A Detection Index for Two Colors

Suppose a binary string contains one substring of length $m$ including $a$ copies of 1, and another of length $m$ including $c$ copies of 1. Then by sliding a window of length $m$ between those two positions, we can always find a substring of length $m$ including $b$ copies of 1 for any $b$ between $a$ and $c$. Cicalese et al.'s [8] index for jumbled pattern matching in binary strings makes use of that fact. It stores, for each substring length $m$, the minimum and maximum numbers of 1s to occur in length-$m$ substrings. This $\mathcal{O}(n)$-space data structure answers detection queries in $\mathcal{O}(1)$ time by checking whether the desired number of 1s is between the stored bounds. If it also records where the minimum and maximum counts occur, then it can do a binary search to answer witnessing queries in $\mathcal{O}(\lg n)$ time.

Fici and Lipták [12] observed that the minimum or maximum number of 1s can only stay the same or increment when we increment $m$. Gagie et al. [13] pointed out that this means we can store the lists of minima and maxima as bitvectors, with 1s indicating increments, and use rank queries to support $\mathcal{O}(1)$ time access, shrinking the detection index to $\mathcal{O}(n)$ bits (rather than words) of space while retaining $\mathcal{O}(1)$ query time.

Gagie et al. also noted that this idea can be generalized to connected graphs, by generalizing the discrete continuity argument used for strings. If in a connected graph with nodes colored black and white there exists a connected subgraph with $m$ nodes of which $a$ are white, and another connected subgraph with $m$ nodes of which $c$ are white, then there must be a connected subgraph with $m$ nodes of which $b$ are white for each $b$ between $a$ and $c$. A sequence of connected subgraphs each with $m$ nodes serves the same purpose as the sliding window in the string case. Therefore, there exist $\mathcal{O}(n)$-bit, $\mathcal{O}(1)$-query-time detection indexes also for jumbled pattern matching in connected graphs with two colors. Building such indexes is NP-hard in general, because it is NP-complete to determine whether there is a connected subgraph with $m$ nodes of which a given number are white, but it takes polynomial time for graphs with bounded tree-width.

Even more generally, suppose we have a hypergraph on $n$ nodes in which each node is black or white and, for each pair of hyperedges $e$ and $e'$ with $m$ nodes each, there is a sequence of hyperedges with $m$ nodes each that starts with $e$ and ends with $e'$ and in which each consecutive pair differs on two nodes. Then there exists a $\mathcal{O}(n)$-bit, $\mathcal{O}(1)$-time detection index for jumbled pattern matching in this hypergraph, although it may not be feasible to build it. The construction time depends on how quickly we can determine the minimum and maximum numbers of white nodes in hyperedges of each size.

In the following lemmas we apply that argument to rectangles on the plane. Given a set of $n$ black and white points in general position on the plane, let them be the nodes of a hypergraph whose hyperedges are the subsets of points that can be contained by axis-aligned rectangles. We first show the existence of a sequence of hyperedges with the necessary continuity property from the minimum to maximum number of white points for each size $m$; that implies the existence of a detection index. We then show how to construct the index efficiently.

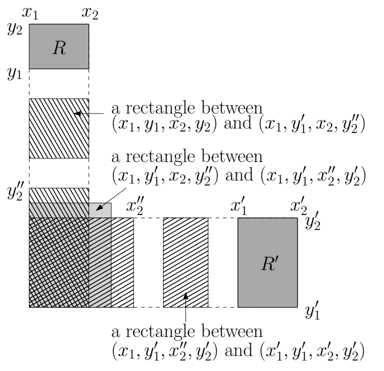
**Lemma 1.** *Given a set of $n$ black and white points in general position in the plane and a pair of axis-aligned rectangles $R$ and $R'$ each containing $m$ points, there exists a sequence of axis-aligned rectangles containing $m$ points each that starts with $R$ and ends with $R'$ and in which each consecutive pair differs on two points.*

*Proof.* We denote an axis-aligned rectangle by an ordered quadruple $(a, b, c, d)$ where $(a, b)$ are the coordinates of the lower left corner and $(c, d)$ are the coordinates of the upper right corner. Consider a set of $n$ black and white points in general position in the plane and a pair of axis-aligned rectangles $R = (x_1, y_1, x_2, y_2)$ and $R' = (x'_1, y'_1, x'_2, y'_2)$ each containing $m$ points. Unless $R$ and $R'$ contain exactly the same subset of points (in which case the lemma holds trivially), neither $R$ nor $R'$ can completely contain the other. Therefore, each rectangle has at least one edge completely outside the other. Up to symmetry, this leaves two cases for us to consider: $x_1 \leq x'_1$ and $y'_1 \leq y_1$; or $x_1 \leq x'_1$ and $x_2 \leq x'_2$ and $y_1 \leq y'_1 \leq y'_2 \leq y_2$.

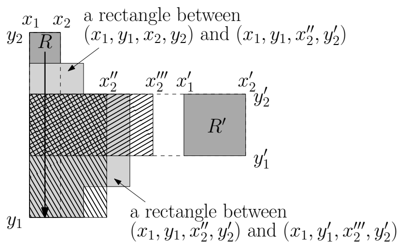
$\mathbf{x_1 \leq x'_1}$ **and** $\mathbf{y'_1 \leq y_1}$**:** Since $R \subseteq (x_1, y'_1, x_2, y_2)$, there exists a rectangle $(x_1, y'_1, x_2, y''_2)$ with $y''_2 \leq y_2$ that contains exactly $m$ points. Similarly, since $R' \subseteq (x_1, y'_1, x'_2, y'_2)$, there exists a rectangle $(x_1, y'_1, x''_2, y'_2)$ with $x''_2 \leq x'_2$ that contains exactly $m$ points. Figure 1a shows an example.

If we can construct three sequences of axis-aligned rectangles containing $m$ points each such that each consecutive pair of rectangles differs on two points and

- one sequence $S_1$ starts with $R = (x_1, y_1, x_2, y_2)$ and ends with $(x_1, y'_1, x_2, y''_2)$,
- one sequence $S_2$ starts with $(x_1, y'_1, x_2, y''_2)$ and ends with $(x_1, y'_1, x''_2, y'_2)$,


$x_1$
$x_2$
$y_2$
$R$
$y_1$
a rectangle between $(x_1, y_1, x_2, y_2)$ and $(x_1, y'_1, x_2, y''_2)$
a rectangle between $(x_1, y'_1, x_2, y''_2)$ and $(x_1, y'_1, x''_2, y'_2)$
$y''_2$
$x''_2$
$x'_1$
$x'_2$
$y'_2$
$R'$
$y'_1$
a rectangle between $(x_1, y'_1, x''_2, y'_2)$ and $(x'_1, y'_1, x'_2, y'_2)$

(a) $x_1 \le x'_1$ and $y'_1 \le y_1$


$x_1$
$x_2$
a rectangle between $(x_1, y_1, x_2, y_2)$ and $(x_1, y_1, x''_2, y'_2)$
$y_2$
$R$
$x''_2$
$x'''_2$
$x'_1$
$x'_2$
$y'_2$
$R'$
$y'_1$
$y_1$
a rectangle between $(x_1, y_1, x''_2, y'_2)$ and $(x_1, y'_1, x'''_2, y'_2)$

(b) $x_1 \le x'_1$; $x_2 \le x'_2$; $y_1 \le y'_1 \le y'_2 \le y_2$

Fig. 1: Suppose there is a pair of axis-aligned rectangles $R = (x_1, y_1, x_2, y_2)$ and $R' = (x'_1, y'_1, x'_2, y'_2)$ each containing $m$ points. We claim there is a sequence of axis-aligned rectangles containing $m$ points each that starts with $R$ and ends with $R'$ and in which each consecutive pair differs on two points.

– one sequence $S_3$ starts with $(x_1, y'_1, x''_2, y'_2)$ and ends with $R' = (x'_1, y'_1, x'_2, y'_2)$,

then by concatenation we can construct another such sequence that starts with $R$ and ends with $R'$.

To construct $S_1$, we start with $R = (x_1, y_1, x_2, y_2)$ and alternately decrease the first y-coordinate until it equals $y'_1$ or a point enters the rectangle, then decrease the second y-coordinate until it equals $y''_2$ or a point leaves the rectangle, and repeat. This way, we keep the number of points within the rectangle the same; since the points are in general position, they enter and leave the rectangle one by one. We construct $S_3$ similarly.

To construct $S_2$, we start with $(x_1, y'_1, x_2, y''_2)$. Assume $x_2 \le x''_2$ and $y'_2 \le y''_2$; the other cases are symmetric. We alternately increase the second x-coordinate until it equals $x''_2$ or a point enters the rectangle, then decrease the second y-coordinate until it equals $y'_2$ or a point leaves the rectangle, and repeat.

$\mathbf{x_1 \le x'_1;\ x_2 \le x'_2;\ y_1 \le y'_1 \le y'_2 \le y_2}$**:** For the sake of brevity, we leave some of the details of this case to the full version of this paper, but Figure 1b shows an illustration. There exist axis-aligned rectangles $(x_1, y_1, x''_2, y'_2)$ and $(x_1, y'_1, x'''_2, y'_2)$ with $x_2 \le x''_2 \le x'''_2 \le x'_2$ that each contain exactly $m$ points.

If we can construct three sequences of axis-aligned rectangles containing $m$ points each such that each consecutive pair of rectangles differ on two points and

– one sequence $S_4$ starts with $R = (x_1, y_1, x_2, y_2)$ and ends with $(x_1, y_1, x''_2, y'_2)$,
– one sequence $S_5$ starts with $(x_1, y_1, x''_2, y'_2)$ and ends with $(x_1, y'_1, x'''_2, y'_2)$,
– one sequence $S_6$ starts with $(x_1, y'_1, x'''_2, y'_2)$ and ends with $R' = (x'_1, y'_1, x'_2, y'_2)$,

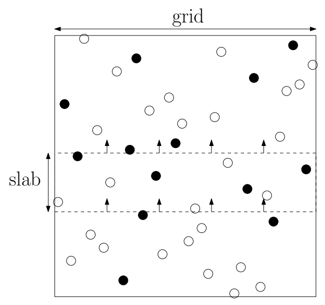


Fig. 2: Sliding a slab of height 9 up a $40 \times 40$ grid. In the previous step, the contents of the slab corresponded to the binary string 110011010. One black point left the slab and another entered it, and the contents of the slab currently correspond to 101011010. In the next step, a white point will leave the slab and a black point will enter it, and the contents of the slab will correspond to 101001010.

then by concatenation we can construct another such sequence that starts with $R$ and ends with $R'$.

We construct $S_4$ and $S_5$ similarly to how we construct sequence $S_2$ as described above. We construct $S_6$ similarly to how we construct $S_1$ and $S_3$. Concatenating all these sequences, the lemma follows. □

Lemma 1 implies the existence of a $\mathcal{O}(n)$-bit space, $\mathcal{O}(1)$-time, detection index for jumbled pattern matching on axis-aligned rectangles with bichromatic points. The index stores the minimum and maximum number of white points using succinct bitvectors, much as in the substring problem. It remains to actually construct that index as quickly as possible. Construction time of $\mathcal{O}(n^4)$ is sufficient by reducing the point set to rank space, building one data structure for $\mathcal{O}(1)$-time range counting for the black points and another for the white points (using $\mathcal{O}(n^2)$ time and space), and checking the number of white and black points in each of the $\mathcal{O}(n^4)$ possible rectangles. We show a tighter bound.

Barba et al. [2] enumerate all possible rectangles by considering slabs of rows in the grid of each height from 1 to $n$, sweeping each of them up from the bottom to the top of the grid. We avoid the need to consider all rectangles individually by maintaining a dynamic list of the points in the current slab and solving a one-dimensional version of the problem within the slab. Figure 2 shows an example.

Dynamically maintaining the point set in the current slab such that we can find the minimum and maximum numbers of white points in rectangles that cover the vertical extent of the slab, is equivalent to maintaining lists of the minimum and maximum numbers of 1s in substrings of each size in a dynamic binary string subject to insertions and deletions. We use the following lemma, whose proof we defer to the full version.

**Lemma 2.** *We can maintain a dynamic binary string of up to $n$ bits in $\mathcal{O}(n)$ space such that inserting or deleting a bit takes $\mathcal{O}(n \lg n)$ time and, given $m$,* *determining the minimum and maximum numbers of 1s in a substring of length $m$ takes $\mathcal{O}(1)$ time.*

Combining Lemmas 1 and 2 gives the construction time of the detection data structure.

**Theorem 2.** *Given a set of $n$ points in the plane colored black and white, in $\mathcal{O}(n^3 \lg n)$ time we can build a $\mathcal{O}(n)$-bit index such that later, given a vector $\boldsymbol{C} = (c_1, c_2)$, in $\mathcal{O}(1)$ time we can determine whether there exists an axis-aligned rectangle containing exactly $c_1$ white points and $c_2$ black points.*

*Proof.* By Lemma 1, there exists an axis-aligned rectangle containing exactly $c_1$ points of the first color and exactly $c_2$ points of the second color if and only if $c_1$ is between the minimum and maximum number of white points in all axis-aligned rectangles containing $m = c_1 + c_2$ points. As in the substring data structure, we need only store the minimum and maximum values of $c_1$ for each value of $m$, and we can do that in the stated space and time bounds.

To construct the data structure, we consider for each possible coordinate for the bottom of a rectangle, every possible coordinate for the top of the rectangle. That gives us a sequence of $\mathcal{O}(n^2)$ slabs, each differing from the previous one by insertion or deletion of one point. Maintaining the data structure of 2 on the string of bits indicating the colors of points in the current slab, we can find the global minimum and maximum numbers of white points for all rectangles in $\mathcal{O}(n^3 \lg n)$ time, and the result follows. □

## 4 A Witnessing Index for Two Colors

The witnessing index of Cicalese et al. [10] for jumbled pattern matching in a binary string is similar to their detection index, but also stores, for every substring length $m$, the positions of two substrings that witness the minimum and maximum number of 1s, as well as storing the string itself as a bitvector. Then a binary search between the stored extrema gives $\mathcal{O}(\lg n)$ query time on the $\mathcal{O}(n)$-space data structure. At each step of the search they perform $\mathcal{O}(1)$ rank queries on the bit vector to determine the number of 1s in the length-$m$ substring at the current position.

The sequence of rectangles described in Lemma 1 makes possible the same kind of binary search on axis-aligned rectangles in bichromatic point sets. Instead of using a bitvector, however, we use a $\mathcal{O}(n)$-space data structure by Brodal et al. [4] that supports two-dimensional three-sided range-selection queries in $\mathcal{O}(\lg n / \lg \lg n)$ time. Given a three-sided range and an integer $k$, this data structure returns the $k$th point in that range, counting away from the middle side. We can use instances of the same data structure to support range counting of the black and white points, with the same query time (although there are faster alternatives [7]).

**Theorem 3.** *Given a set of $n$ black and white points in the plane, in $\mathcal{O}(n^3 \lg n)$ time we can build in $\mathcal{O}(n^3 \lg n)$ time a $\mathcal{O}(n)$-space index such that later, given* *a vector $\boldsymbol{C} = (c_1, c_2)$, in $\mathcal{O}(\lg^2 n / \lg\lg n)$ time we can return an axis-aligned rectangle containing exactly $c_1$ white points and exactly $c_2$ black points, if one exists.*

*Proof.* We use the same quadruple notation for rectangles as in the proof of Lemma 1, and as in Theorem 2, we build the data structure in $\mathcal{O}(n^3 \lg n)$ time by examining horizontal slabs to find the global maximum and minimum number of white points for rectangles with each value of $m = c_1 + c_2$. We record the coordinates of the extremal rectangles as well as their white point counts.

We do not include a figure specifically for this proof; instead, we refer the reader back to Figure 1a.

For each $m$, we record whether $R = (x_1, y_1, x_2, y_2)$ and $R' = (x'_1, y'_1, x'_2, y'_2)$ are symmetric to the first case we considered in Lemma 1 (i.e., $x_1 \leq x'_1$ and $y'_1 \leq y_1$) or symmetric to the second case (i.e., $x_1 \leq x'_1$ and $x_2 \leq x'_2$ and $y_1 \leq y'_1 \leq y'_2 \leq y_2$). In this version of this paper, we describe how to deal only with the first case; the second is similar.

Suppose $x_1 \leq x'_1$ and $y'_1 \leq y_1$. Then we also record the positions of the intermediate rectangles $(x_1, y'_1, x_2, y''_2)$ and $(x_1, y'_1, x''_2, y'_2)$ described for the this case in Lemma 1. Finally, we record the number of white points in each of $R$, $(x_1, y'_1, x_2, y''_2)$, $(x_1, y'_1, x''_2, y'_2)$ and $R'$.

Let $S_1$, $S_2$ and $S_3$ again be the sequences of axis-aligned rectangles containing $m$ points each that we described for this case. We can determine from the number of white points in $R$, $(x_1, y'_1, x_2, y''_2)$, $(x_1, y'_1, x''_2, y'_2)$ and $R'$ whether we should binary search in $S_1$, $S_2$ or $S_3$. Searching in $S_3$ is analogous to searching in $S_1$.

To perform binary search in $S_1$—i.e., in the sequence of axis-aligned rectangles between $R = (x_1,\ y_1,\ x_2,\ y_2)$ and $(x_1, y'_1, x_2, y''_2)$ each containing $m$ points, with $y'_1 \leq y_1$—we perform a binary search in the range $[y'_1, y_1]$. For the sake of simplicity, we assume we reduce all coordinates to rank space when we build the index, so $y'_1$ and $y_1$ are integers that differ by at most $n$.

At each step of the binary search, we choose some integer $y'''_1$ between the endpoints of our current range of integers, as the first y-coordinate of the rectangle we will test. We use a range-selection query to find the $m$th point from the bottom of the range $(x_1, y'''_1, x_2, \infty)$, which tells us the second y-coordinate $y'''_2$ of that test rectangle. We then count the number of white points in $(x_1, y'''_1, x_2, y'''_2)$, which tells us on which half of the current range we should recurse. We use $\mathcal{O}(\lg n / \lg\lg n)$ time for each step of the binary search, so $\mathcal{O}(\lg^2 n / \lg\lg n)$ time overall.

To perform binary search in $S_2$—i.e., in the sequence of axis-aligned rectangles between $(x_1, y'_1, x_2, y''_2)$ and $(x_1, y'_1, x''_2, y'_2)$ each containing $m$ points—we perform binary search in the range $[x_2, x''_2]$. Again, we assume $x_2 \leq x''_2$ and $y'_2 \leq y''_2$, because the other cases are symmetric.

At each step of the binary search, we choose some integer $x'''_2$ between the endpoints of our current range of integers, as the second x-coordinate of the rectangle we will test. We use a range-selection query on to find the $m$th point from the bottom of the range $(x_1, y'_1, x'''_2, \infty)$, which tells us the second y-coordinate $y'''_2$ of that test rectangle. We then count the number of white points in $(x_1, y'_1, x'''_2, y'''_2)$, which tells us on which half of the current range we should recurse. Again, we use $\mathcal{O}(\lg n / \lg\lg n)$ time for each step of the binary search, so $\mathcal{O}(\lg^2 n / \lg\lg n)$ time overall. $\square$

**Acknowledgments**

Many thanks to the organizers and participants of CCCG 2013 and Stringmasters 2013, especially L. Barba, F. Cicalese, S. Denzumi, M. He, J. Holub, J. Kärkkäinen, A. Kawamura, D. Kempa, T. Kociumaka, Z. Lipták, J. Tarhio and G. Zhou; and to E. Giaquinta, M. Lewenstein, R. Rizzi and A. I. Tomescu.

## References

1. Badkobeh, G., Fici, G., Kroon, S., Lipták, Z.: Binary jumbled string matching for highly run-length compressible texts. IPL **113** (2013) 604–608
2. Barba et al., L.: On $k$-enclosing objects in a coloured point set. In: Proc. CCCG. (2013) 229–234
3. Björklund, A., Kaski, P., Kowalik, L.: Probably optimal graph motifs. In: Proc. STACS. (2013) 20–31
4. Brodal, G.S., Gfeller, B., Jørgensen, A.G., Sanders, P.: Towards optimal range medians. TCS **412** (2011) 2588–2601
5. Burcsi, P., Cicalese, F., Fici, G., Lipták, Z.: Algorithms for jumbled pattern matching in strings. IJFCS **23** (2012) 357–374
6. Butman, A., Eres, R., Landau, G.M.: Scaled and permuted string matching. IPL **92** (2004) 293–297
7. Chan, T.M., Wilkinson, B.T.: Adaptive and approximate orthogonal range counting. In: Proc. SODA. (2013) 241–251
8. Cicalese, F., Fici, G., Lipták, Z.: Searching for jumbled patterns in strings. In: Proc. PSC. (2009) 105–117
9. Cicalese, F., Laber, E.S., Weimann, O., Yuster, R.: Near linear time construction of an approximate index for all maximum consecutive sub-sums of a sequence. In: Proc. CPM. (2012) 149–158
10. Cicalese et al., F.: Indexes for jumbled pattern matching in strings, trees and graphs. In: Proc. SPIRE. (2013) 56–63
11. Fellows, M.R., Fertin, G., Hermelin, D., Vialette, S.: Upper and lower bounds for finding connected motifs in vertex-colored graphs. JCSS **77** (2011) 799–811
12. Fici, G., Lipták, Z.: On prefix normal words. In: Proc. DLT. (2011) 228–238
13. Gagie, T., Hermelin, D., Landau, G.M., Weimann, O.: Binary jumbled pattern matching on trees and tree-like structures. In: Proc. ESA. (2013) 517–528
14. Giaquinta, E., Grabowski, S.: New algorithms for binary jumbled pattern matching. IPL **113** (2013) 538–542
15. Kociumaka, T., Radoszewski, J., Rytter, W.: Efficient indexes for jumbled pattern matching with constant-sized alphabet. In: Proc. ESA. (2013) 625–636
16. Lacroix, V., Fernandes, C.G., Sagot, M.F.: Motif search in graphs: Application to metabolic networks. TCBB **3** (2006) 360–368
17. Moosa, T.M., Rahman, M.S.: Sub-quadratic time and linear space data structures for permutation matching in binary strings. JDA **10** (2012) 5–9