

Color-Spanning Problem for Line Segments

Sukanya Maji(✉) and Sanjib Sadhu

Department of CSE, National Institute of Technology Durgapur, Durgapur, India
sm.20cs1102@phd.nitdgp.ac.in, sanjib.sadhu@cse.nitdgp.ac.in

Abstract. The color-spanning problem for a given set of colored geometric objects finds a region that contains at least one object of each color. This paper studies the color-spanning problem for a given set of colored line segments $\mathcal{L}$ lying in $\mathbb{R}^2$. For n line segments, each associated with any of m colors ($3 \leq m < n$), we compute (i) one (resp. two congruent disjoint) color-spanning vertical strip (resp. strips) of minimum width in $O(n)$ time after $O(n \log n)$ preprocessing task, (ii) one (resp. two congruent disjoint) color-spanning axis parallel square (resp. squares) of minimum perimeter in $O(n(n-m))$ (resp. $O(n^3)$) time. The same results also exist if the inputs to the aforesaid problems are n simple polygons or disks. Our algorithm improved the time complexity of finding the minimum perimeter color-spanning square problem for the segments [1] by $\log n$ factor. Finally, we show the $\sqrt{2}$ approximation algorithm for computing one (resp. two congruent disjoint) color-spanning disk (resp. disks) of minimum radius for $\mathcal{L}$.

Keywords: Computational geometry · Color-spanning objects · Line sweeping · Approximation algorithm

1 Introduction

We are given m different types (e.g. hospitals, post-offices, schools, shopping malls, fire-stations etc.) of n resources, where $3 \leq m < n$. These resources are generally represented by geometric objects e.g. points, line segments, simple polygons, circular disks etc. in $\mathbb{R}^2$. The same (resp. different) types of resources are identified by assigning the same (resp. different) colors to them. A color-spanning region contains at least one resource of each color, where the region may be of an unbounded (e.g. a strip) or bounded (e.g. a disk, rectangle, square etc.) types. The *color-spanning problem* computes the smallest color-spanning region for the given set of resources and it was first introduced in 2001 [2], where only colored points were considered as input. The color-spanning problem for points has grown a lot of interest to the researchers due to its huge applications in the facility location problem [2], pattern recognition [4], database queries [10] etc.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022
O. Gervasi et al. (Eds.): ICCSA 2022, LNCS 13375, pp. 76–89, 2022.
https://doi.org/10.1007/978-3-031-10522-7_6

Related Work: A lot of research works have been done on a set of colored points. Huttenlocher et al. [1] computed the smallest color-spanning circle for a given set of n points with m colors in $O(mn \log n)$ time. Abellanas et al. [2] first computed the color-spanning axis-parallel rectangle for point set in $O(n(n-m)\log^2 m)$. The smallest color-spanning strip and rectangle of arbitrary orientation for a given set of points can be computed in $O(n^2 \log n)$ and $O(n^3 \log m)$ time [3], respectively. The color-spanning axis-parallel square and axis-parallel equilateral triangle can be determined in $O(n \log^2 n)$ [5] and $O(n \log n)$ [6] time, respectively. Acharyya et al. [11] identified the smallest color-spanning axis-parallel square, rectangle and circle for a point set that must contain also a given query point q. The minimum width color-spanning circular annulus, and axis-parallel rectangular annulus and axis-parallel square annulus for a set of points can be computed in $O(n^3 \log n)$, $O(n^4)$ and $O(n^3 + n^2 m \log m)$ time [9], respectively. Sang Won Bae [7] also computed the minimum width color-spanning axis-parallel rectangular annulus in $O((n-m)^3 n \log n)$ time. Most of the research works on color-spanning problem deal with only point set as input. However, in real perspective it is not always reasonable to represent each facility by point only. Rather, depending on the size and shape of the facilities, they are represented more precisely by the line segments, simple polygons or disks instead of points. So, in this paper we are interested in the color-spanning objects which cover a set of line segments, simple polygons or disks. Also, the solution to the color-spanning problem for point set cannot be applied to the color-spanning problem for line segments. To the best of our knowledge, there exists one work on color-spanning problem for line segments by Huttenlocher et al. [1], where they first computed the upper envelope of the Voronoi surfaces of the line segments and using that envelope, they shown how to compute the smallest color-spanning axis-parallel square (resp. disk) of the line segments, in $O(n^2 \log n)$ (resp. $O(n^2 \alpha(n) \log n)$) time. In our work, we have improved the time complexity of finding the color-spanning axis-parallel square [1] by $\log n$ factor using the line sweeping technique. We have also studied the color-spanning problem where colored segments are covered by two congruent disjoint strips or squares of minimum size. This is motivated by a variation of color-spanning problem, where for a given set of resources (colored points or segments), k congruent color-spanning objects of minimum size is to be determined, with the constraint that each resource r_i can support at most $f(r_i)$ demand points (centers of the color-spanning regions). For simplicity, we have taken $k = 2$ and $f(r_i) = 1$ for each resource r_i.

Our Contributions: We propose algorithm(s) to compute color-spanning vertical strip(s) or axis parallel square(s) of minimum size for n line segments or polygons or disks in $\mathbb{R}^2$.

Color-spanning problems for line segments	Time	Space	Exact/Approx
One color-spanning vertical strip of minimum width	$O(n)$ with $O(n \log n)$ preprocessing	$O(n)$	Exact
Two congruent color-spanning disjoint vertical strips of minimum width	$O(n)$ with $O(n \log n)$ preprocessing	$O(n)$	Exact
One color-spanning axis-parallel square of minimum perimeter	$O(n(n-m))$	$O(n)$	Exact
Two congruent disjoint color-spanning axis-parallel squares of minimum perimeter	$O(n^3)$	$O(n^2)$	Exact
One color-spanning disk of minimum radius	$O(n(n-m))$	$O(n)$	$\sqrt{2}$-factor approximation
Two congruent disjoint color-spanning disks of minimum radius	$O(n^3)$	$O(n^2)$	$\sqrt{2}$-factor approximation

2 Preliminaries and Notations

Definition 1 (Covering). *A geometric object in $\mathbb{R}^2$ is said to be* covered *by a region on that plane, if each point of that object lies inside or on the boundary of the region.*

Definition 2 (Color-spanning region). *Let there be n geometric objects e.g. points, line segments in $\mathbb{R}^2$, where each object is associated with one of the m distinct colors ($3 \leq m < n$). A region in $\mathbb{R}^2$ is said to be* color-spanning*, if it covers at least one object of each color.*

Definition 3 (Vertical strip). *A vertical strip V is an unbounded region enclosed by two vertical lines, $left(V)$ and $right(V)$ which are the left and right boundaries of V, respectively.*

The x-coordinate and y-coordinate of a point p are denoted by $x(p)$ and $y(p)$, respectively. The width of a vertical strip V is determined by the distance between its two boundaries. The given set of line segments are denoted by $\mathcal{L} = \{\ell_1, \ell_2, \ldots, \ell_n\}$. We use four arrays $\mathcal{L}_\ell$, $\mathcal{L}_r$, $\mathcal{L}_t$ and $\mathcal{L}_b$, each with the members in $\mathcal{L} = \{\ell_1, \ell_2, \ldots, \ell_n\}$ sorted with respect to their left, right, top, and bottom endpoints, respectively. Each element $\ell_i \in \mathcal{L}$ maintains four pointers to the corresponding element in $\mathcal{L}_\ell$, $\mathcal{L}_r$, $\mathcal{L}_t$ and $\mathcal{L}_b$. Also, each element of $\mathcal{L}_i$, $i = \ell, r, t, b$ points to the corresponding segment $\ell_i \in \mathcal{L}$. The arrays $\mathcal{L}_i$, $i = l, r, t, b$ can be created in $O(n \log n)$ time and will be stored using $O(n)$ space. We also maintain another array $\mathcal{C} = (c_1, c_2, \ldots, c_m)$ for m distinct colors to keep the count of segments of each color, where $c_i \in \mathbb{Z}^+$ for each $1 \leq i \leq m$. The array $\mathcal{C}$ is initialized with zero. Each color is also represented by distinct integer $w \in \{1, 2, \ldots, m\}$.

3 Color-Spanning Vertical Strip

In this section, three problems are discussed depending on the number (one or two) of color-spanning regions and type of inputs (line segments, simple polygons and disks).

3.1 Color-Spanning Problem Using One Vertical Strip

Problem 1. *Given a set of n line segments $\mathcal{L} = \{\ell_1, \ell_2, \ldots, \ell_n\}$ in $\mathbb{R}^2$ and m distinct colors ($3 \leq m < n$), where each line segment is associated with one of such m colors. The objective is to compute a color-spanning vertical strip V of minimum width.*

Definition 4 (*minimal-CSVS*). *A color-spanning vertical strip that cannot be shrunk further without violating the definition of color-spanning region, is said to be minimal width color-spanning vertical strip (**minimal-CSVS**).*

Definition 5 (*minimum-CSVS*). *There may exist more than one minimal-CSVS for a set of segments in $\mathcal{L}$. The one having minimum width among all such minimal-CSVSs is said to be minimum width color-spanning vertical strip (**minimum-CSVS**).*

Observation 1. *For any minimal-CSVS, say V, its $left(V)$ (resp. $right(V)$) must pass through the left endpoint (resp. right endpoint) of a segment, say ℓ_i (resp. ℓ_j). We call ℓ_i and ℓ_j as the defining segments for the $left(V)$ and $right(V)$ of the strip V, respectively.*

Property 1 (Property of *minimal-CSVS*). *Suppose V is a minimal-CSVS. The color of the line segments that define the $left(V)$ and $right(V)$ of the strip V, must be different and unique among the segments lying inside the V, otherwise, V will contain another minimal-CSVS inside it as a proper subset contradicting the assumption that V is a minimal-CSVS.*

Algorithm: We compute all possible *minimal-CSVS*s by sweeping two vertical lines L_1 and L_2 from left to right through the left endpoint and right endpoint of the segments in the arrays $\mathcal{L}_\ell$ and $\mathcal{L}_r$, respectively (see Observation 1), as follows:

First, we set L_1 and L_2 to the left endpoint and right endpoint of the first segment in array $\mathcal{L}_\ell$ and $\mathcal{L}_r$, respectively. Then sequentially move L_2 rightward as mentioned above. As the sweep lines move, we update the array $\mathcal{C}$ to reflect the color of the segments lying within the strip bounded by L_1 and L_2. The L_2 stops when the value of each member of array $\mathcal{C}$ becomes at least 1. Now, if the color of the segment defining L_1 repeats more than once within the strip bounded by L_1 and L_2, then we shift L_1 rightward until the Property 1 of a *minimal-CSVS* is satisfied, and compute the width of this strip. We repeat the above procedure by shifting L_1 rightward to its next segment to obtain a new *minimal-CSVS*. This process is continued until L_1 reaches to the $(n-m)^{th}$ segment, since a

minimal-CSVS will cover at least m segments. We compute all such *minimal-CSVS*s, and finally report the one with smallest width as *minimum-CSVS*. We need linear sized arrays $\mathcal{C}$, $\mathcal{L}_\ell$, $\mathcal{L}_r$ to compute the *minimum-CSVS*. In each step of the above procedure, either L_1 or L_2 moves rightward and one segment is processed in each step. Thus we obtain the following result.

Theorem 1. *We can compute the minimum-CSVS of a set of n line segments in $O(n)$ time and $O(n)$ space, after $O(n \log n)$ preprocessing task.*

3.2 Color-Spanning Problem Using Two Vertical Strips

Problem 2. *Given a set of n line segments $\mathcal{L} = \{\ell_1, \ell_2, \ldots, \ell_n\}$ in $\mathbb{R}^2$, where each segment ℓ_i is associated with one of the m distinct colors ($3 \leq m < n$). The objective is to compute two congruent color-spanning disjoint vertical strips V_1 and V_2 of minimum width.*

This problem is equivalent to compute a pair of *minimal-CSVS* (V_1, V_2) so that the width of its larger strip is minimized among the larger strips of all possible pair of *minimal-CSVS*s. For each *minimal-CSVS* V_1 of the line segments in $\mathcal{L}$, we choose the *minimum-CSVS* V_2 of the segments that lie to the right of V_1, as follows:

We compute all the *minimal-CSVS*s of the segments in $\mathcal{L}$ by sweeping L_1 and L_2 from right to left, i.e. in the reverse order as described in the Sect. 3.1 for the Problem 1. We also assign a strip-id to each of such strip sequentially starting with 1 in the order of their computations i.e. the strip-id of the rightmost *minimal-CSVS* is 1. We construct two arrays $BOUNDARY$ and $STRIP$, while computing these *minimal-CSVS*s.

(i) The array $BOUNDARY$ has three fields - $left_boundary$, $size$ and $strip_id$. If a *minimal-CSVS* V is also a *minimum-CSVS* for the segments that lie to the right of $left(V)$, then we store the $left(V)$ in the $left_boundary$ field of the array $BOUNDARY$. The size and the strip-id of this *minimum-CSVS* are stored in the fields $size$ and $strip_id$ of the array $BOUNDARY$, respectively. The first entry of the array $BOUNDARY$ stores the information for the rightmost *minimal-CSVS*.

(ii) The array $STRIP$ has 3 fields: $left_boundary$, $right_boundary$ and $strip_id$ which stores the $left_boundary$, $right_boundary$ and the $strip_id$ of each *minimal-CSVS*, respectively.

The information of all the *minimal-CSVS*s are inserted sequentially into the array $STRIP$ as they are computed, i.e. the first entry (resp. last entry) in the array $STRIP$ corresponds to the leftmost (resp. rightmost) *minimal-CSVS*. However, in case of array $BOUNDARY$, while computing the ith *minimal-CSVS* (i.e. with $strip_id$ i), we append its $left_boundary$, $size$, and $strip_id$ in the array $BOUNDARY$, provided its width, say w', is less than the width, say w, of the *minimum-CSVS* obtained so far. Here the w can be obtained from the last row (inserted so far) of the array $BOUNDARY$.

It is to be noted that the rows in the array $BOUNDARY$ are inserted in the descending order with respect to the x-coordinate of the $left_boundary$ field. For each *minimal-CSVS*, say V_i, stored in the array $STRIP$, we need to search the array $BOUNDARY$ for the *minimum-CSVS* that lie to the right of V_i. To accomplish this, we search both these arrays in upward direction starting from their bottom row as follows:

In each iteration, for a strip with *strip_id*, say V_i, stored in the array $STRIP$, we search the array $BOUNDARY$ in upward direction, until we get a strip, say V_j, so that $left(V_j) \geq right(V_i)$, and then we stop to get a pair (V_i, V_j). Note that the V_j is the *minimum-CSVS* for the segments lying to the right of *minimal-CSVS* V_i. In the next iteration, we consider the strip with *strip_id* V_{i-1} stored in the previous row of the array $STRIP$. Now, we find the *minimum-CSVS* for the segments lying to the right of V_{i-1} and we search the array $BOUNDARY$ from the row where searching was stopped in previous iteration. We repeat the above steps to compute all possible pairs (V_i, V_j) in linear amount of (amortized) time, where V_j is the *minimum-CSVS* for the segments that lie to the right of the *minimal-CSVS* V_i. We compare the size of the larger strips of each pair (V_i, V_j) to choose the minimum one and report that pair. If there are no such pair of disjoint *minimal-CSVS*s for the given segments in $\mathcal{L}$, then also we can report it in $O(n)$ time. In this case, for the leftmost $CSVS$, say V, there does not exist a *minimum-CSVS* to the right of $right(V)$. Thus we obtain the following result.

Theorem 2. *For a given set of n line segments in $\mathbb{R}^2$, we can compute two congruent color-spanning vertical strips of the minimum width, if such a pair exists, otherwise we report that no such pair exists in $O(n)$ time and $O(n)$ space, after $O(n \log n)$ preprocessing.*

3.3 Color-Spanning Vertical Strip for a Set of Simple Polygons or Disks

If the inputs to the Problem 1 and Problem 2 are colored simple polygons or disks (instead of line segments), then to determine the color-spanning vertical strip(s) of minimum width for these objects, we replace each of these simple polygons (resp. disks) with the line segments (i.e. each polygon/disk corresponds to a segment) in such a way that if a vertical strip exactly[1] covers, k (say) such line segments, then that vertical strip also covers exactly their corresponding k polygons (resp. disks) and vice versa. The color of these line segments are kept same as that of the given simple polygons or disks. We do this as follows:

If the inputs is a simple polygon, we replace it by a segment that connects its extreme leftmost and rightmost vertices, whereas for disk, we replace it by its horizontal diameter. Afterwards, we apply the algorithm of the Problem 1 and Problem 2 to solve the respective color-spanning problem for simple polygons or disks. So, we get the following result.

[1] A polygon is said to be exactly covered by a region, if we reduce the size of the region then at least one point of the polygon will lie outside the region.

Theorem 3. *For a given set of n simple polygons/disks, we can compute one (resp. two congruent) color-spanning vertical strip (resp. disjoint strips) of minimum width in $O(n)$ time and $O(n)$ space after $O(n \log n)$ preprocessing task.*

4 Color-Spanning Axis-Parallel Square

In this section, we discuss the color-spanning problem for the line segments where the color-spanning region is axis-parallel square. Unless otherwise stated, we use square to imply axis-parallel square throughout the paper. We abbreviate color spanning square by CSS. The *size* of a square is defined by its perimeter. We consider three problems in this Section.

4.1 Color-Spanning Problem Using One Axis-Parallel Square

Problem 3. *Given a set of n line segments $\mathcal{L} = \{\ell_1, \ell_2, \ldots, \ell_n\}$ in $\mathbb{R}^2$ and m distinct colors ($3 \leq m < n$), where each line segment is associated with one of such m colors. The objective is to compute a minimum sized color-spanning square $\mathcal{S}$.*

Definition 6 (*minimal-CSS*). *A color-spanning square $\mathcal{S}$ for a given set of line segments, is said to be minimal if $\mathcal{S}$ cannot be shrunk without violating the color-spanning property. This is denoted by **minimal-CSS**.*

Definition 7 (*minimum-CSS*). *Among all possible minimal-CSS for $\mathcal{L}$, the one with smallest size is called minimum color-spanning square and is denoted by **minimum-CSS**.*

Fact 1. *A square $\mathcal{S}$ can be uniquely defined by three points that lie on any three sides of it. However, as a special case, the square $\mathcal{S}$ may also be uniquely identified by two points that are the two endpoints of a diagonal of it.*

We generate the squares with the endpoints of the segments in $\mathcal{L}$. The segments whose endpoint define a square $\mathcal{S}$ uniquely (see Fact 1) are called the "defining segments" for $\mathcal{S}$.

Fact 2. *If $\ell_i \in \mathcal{L}$ defines the left (resp. right) side of a square $\mathcal{S}$, then the left (resp. right) endpoint of ℓ_i lies on that side of $\mathcal{S}$. Similarly, if ℓ_i defines the top (resp. bottom) side of $\mathcal{S}$, then the top (resp. bottom) endpoint of ℓ_i lies on that side of $\mathcal{S}$.*

Depending on the three sides of the square $\mathcal{S}$ on which the endpoint of its defining segments lies, we classify the squares into 4 types: (i) (Left, Top, Right), (ii) (Left, Bottom, Right), (iii) (Left, Top, Bottom) and (iv) (Right, Top, Bottom), where the terms "Left", "Top", "Right" and "Bottom" denote the side of the square $\mathcal{S}$ on which an endpoint of the respective defining segment of $\mathcal{S}$ lies. We first generate all possible squares whose left side are defined by a segment, say $\ell_i \in \mathcal{L}$ (see Fact 2). In other words, we compute the squares of types (i), (ii) and (iii) as

mentioned above, with ℓ_i as the defining segment of their left side. These types of squares are generated using a sweep line L, and two sweep rays $\mathcal{R}_1$ and $\mathcal{R}_2$, as follows:

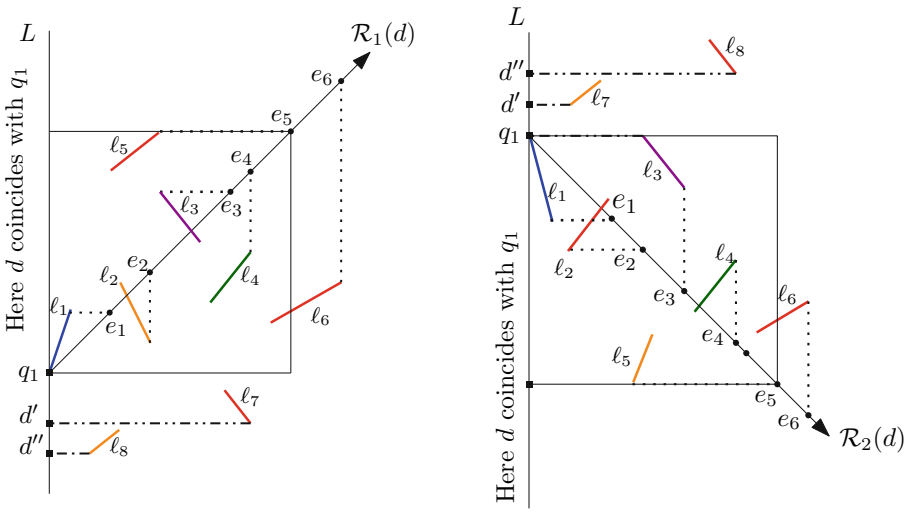
(i) The line L is vertical that sweeps rightwards through its event points that are the left endpoints of the segments stored in the sorted array $\mathcal{L}_\ell$ (defined in Sect. 2).
(ii) The sweep ray $\mathcal{R}_1$ (resp. $\mathcal{R}_2$) is a transverse ray of positive (resp. negative) unit slope directed towards north-east (resp. south-east), and it will move vertically downwards (resp. upwards) through its discrete event points (to be discussed later) on the line L. The ray $\mathcal{R}_1$ (resp. $\mathcal{R}_2$) is the locus of the top-right (resp. bottom-right) corner of the squares whose left side and bottom (resp. top) side are defined by the segment $\ell_i \in \mathcal{L}$ and $\ell_j \in \mathcal{L}$, respectively.

Suppose the sweep line L passes through the left endpoint, say q_1, of a segment $\ell_i \in \mathcal{L}$ (see Fig. 1(a)). Consider each segment $\ell_j \in \mathcal{L}$ that lie completely to the right of L. If bottom (resp. top) endpoint of such segments lie below (resp. above) q_1, then from that endpoint take a horizontal projection on the line L, which will be the event point for the sweep ray $\mathcal{R}_1$ (resp. $\mathcal{R}_2$) on L. Note that if slope of ℓ_i is positive (resp. negative), then we take the endpoint q_1 also as an event point for $\mathcal{R}_1$ (resp. $\mathcal{R}_2$). The sorted list of the event points of $\mathcal{R}_1$ (resp. $\mathcal{R}_2$) on L can be generated from the sorted array $\mathcal{L}_b$ (resp. $\mathcal{L}_t$) in $O(n)$ time.

For each different positions (i.e. event points) of L, there is an event point of $\mathcal{R}_1$ (resp. $\mathcal{R}_2$) on L, known as its *"first event point"* from which the ray $\mathcal{R}_1$ (resp. $\mathcal{R}_2$) starts sweeping downwards (resp. upwards). We denote this event point by d whose y-coordinate is largest (resp. smallest) among all the event points of $\mathcal{R}_1$ (resp. $\mathcal{R}_2$) on the line L. The sweep ray $\mathcal{R}_1$ (resp. $\mathcal{R}_2$) emanating from its event point, say p, is represented by $\mathcal{R}_1(p)$ (resp. $\mathcal{R}_2(p)$). We generate the event points of other segments on the ray $\mathcal{R}_1(d)$ (resp. $\mathcal{R}_2(d)$), where these event points indicate the top-right (resp. bottom-right) corner of the squares $\mathcal{S}$ whose bottom-left (resp. top-left) corner is at d.

Computation of Event Points on $\mathcal{R}_1(d)$ and $\mathcal{R}_2(d)$: We denote the two endpoints of a segment $\ell_k \in \mathcal{L}$ by $\ell_k.1$ and $\ell_k.2$. For each segment $\ell_k \in \mathcal{L}$ lying to the right of L, we create an event point e_k on the sweep ray $\mathcal{R}_1(d)$, where e_k satisfies the equation of the ray $\mathcal{R}_1(d)$: $y = x + y(d) - x(d)$. Depending on position of ℓ_k, the events e_k are generated as follows:

(i) ℓ_k lies above $\mathcal{R}_1(d)$: Here $y(e_k) = \max(y(\ell_k.1), y(\ell_k.2))$ (e.g. e_5 due to ℓ_5 in Fig. 1(a)).
(ii) ℓ_k lies below $\mathcal{R}_1(d)$: Here $x(e_k) = \max(x(\ell_k.1), x(\ell_k.2))$ (e.g. e_4 due to ℓ_4 in Fig. 1(a)).
(iii) ℓ_k intersects with $\mathcal{R}_1(d)$: Let the endpoints of ℓ_k lying above and below $\mathcal{R}_1(d)$ be $\ell_k.1$ and $\ell_k.2$, respectively. We take the horizontal (resp. vertical) projection, say p (resp. q) of $\ell_k.1$ (resp. $\ell_k.2$) on $\mathcal{R}_1(d)$. If q lies above p on $\mathcal{R}_1(d)$, then $e_k = q$ (e.g. e_2 due to ℓ_2 in Fig. 1(a)), otherwise $e_k = p$ (e.g. e_3 due to ℓ_3 in Fig. 1(a)).



(a) Event points e_k on sweep ray $\mathcal{R}_1(d)$

(b) Event points e_k on sweep ray $\mathcal{R}_2(d)$

Fig. 1. Generation of event points on the sweep rays $\mathcal{R}_1$ and $\mathcal{R}_2$

In each of the aforesaid cases, once we compute $x(e_k)$ (resp. $y(e_k)$), the corresponding $y(e_k)$ (resp. $x(e_k)$) can be obtained from the equation of $\mathcal{R}_1(d)$ stated above.

Similarly, for each segment $\ell_k \in \mathcal{L}$, we create an event point e_k on the sweep ray $\mathcal{R}_2(d)$ satisfying the equation: $y + x = y(d) + x(d)$ as shown in Fig. 1(b). We denote the color of a line segment $\ell_k \in \mathcal{L}$ by $color(\ell_k)$. We maintain an array $ACTIVE$, each with the members in $\mathcal{L}$, that keeps track of the segments completely covered by the square generated at each event point of $\mathcal{R}_1$ (resp. $\mathcal{R}_2$) on L. We put 1 (resp. 0) to $ACTIVE[k]$, if the segment ℓ_k is covered (resp. not covered) by the square. As mentioned in the Sect. 2, the array $\mathcal{C}$ of size m keeps records of the number of different colors of the line segments that are being covered by the square with its bottom-left (resp. top-left) corner at each event point of $\mathcal{R}_1$ (resp. $\mathcal{R}_2$) on L. Both these arrays are initialized to zeros. To keep track of the number of colors of the segments that are missing inside a square, we take a variable $\#missing_color$ which is initialized to m. For a color-spanning square, $\#missing_color$ is 0. A segment lying inside a square $\mathcal{S}$ is said to be *essential*, if it is the only segment with its color inside $\mathcal{S}$. We also take two variables *topmost* and *rightmost*, where *topmost* (resp. *rightmost*) points to a segment whose top (resp. right) endpoint has largest y-coordinate (resp. x-coordinate) among the other segments lying inside $\mathcal{S}$. As the sweep ray $\mathcal{R}_1$ (resp. $\mathcal{R}_2$) moves down (resp. up) the arrays $ACTIVE$ and $\mathcal{C}$ are dynamically updated.

We denote a square by $\mathcal{S}(d, z)$, where d and z are its bottom-left (resp. top-left) corner and top-right (resp. bottom-right) corner, respectively. We first consider ray $\mathcal{R}_1(d)$, where d is the "*first event point*" of $\mathcal{R}_1$ on L. We compare all the event points e_k generated on $\mathcal{R}_1(d)$ by the segments l_k (discussed earlier) and choose the one with smallest x-coordinate, say $z = e_z$, such that the square $\mathcal{S}(d, z)$ is a CSS. Obviously, this $\mathcal{S}(d, z)$ is of minimum sized among all possible CSS with the bottom-left corner at d. The segments lying inside the square $\mathcal{S}(d, z)$, and their colors are used to update the arrays $ACTIVE$ and $\mathcal{C}$, respectively. Then the sweep ray $\mathcal{R}_1(d)$ moves downward to its next event point, say d', on L and we denote the ray by $\mathcal{R}_1(d')$. Now construct a square $\mathcal{S}(d', z')$, where the point z' on $\mathcal{R}_1(d')$ is vertically below the point z. So, the square $\mathcal{S}(d', z')$ is a congruent to $\mathcal{S}(d, z)$ and lies vertically below it. Suppose the segment ℓ'_j generated the event d' on L. If an endpoint of ℓ'_j lies outside $\mathcal{S}(d', z')$, then we discard $\mathcal{R}_1(d')$, and move $\mathcal{R}_1$ to the next event point, say d'' on L since the CSS with bottom-left corner at d' will be larger than $\mathcal{S}(d, z)$; otherwise, we update the array $ACTIVE$ by (i) inserting ℓ'_j and (ii) deleting the segments from $\mathcal{S}(d, z)$ whose at least one endpoint lies outside $\mathcal{S}(d', z')$. This deletion is done sequentially from $\mathcal{S}(d, z)$ in the decreasing order with respect to their top endpoint. During such insertion and deletion operations, we update (if required) both *topmost* and *rightmost* variable after each such operation. During the deletion operation, if we encounter the segment ℓ_i, we can not delete it, since we are generating the squares whose left side is to be defined by ℓ_i. In that case, we stop and return the square $\mathcal{S}(d, z)$ as a *minimal-CSS* for all the event points of $\mathcal{R}_1$ lying on L. At each operation of insertion and deletion, we update the $\mathcal{C}$ array also as follows:

While deleting (resp. inserting) the segment ℓ_k from (resp. into) the array $ACTIVE$, we decrease (resp. increase) the value of $\mathcal{C}[color(\ell_k)]$ by 1. If the value of an index of C array changes from 1 (resp. 0) to 0 (resp. 1), we increase (resp. decrease) the variable $\#missing_color$ by 1. While generating the square at $\mathcal{R}_1(d)$, if $\#missing_color > 0$ after the completion of the aforesaid update operation, then the square $\mathcal{S}(d', z')$ is not a CSS and we do not compute the *minimal-CSS* for $\mathcal{R}_1(d')$ as it would be larger than the already computed CSS $\mathcal{S}(d, z)$. Hence, $\mathcal{R}_1(d')$ moves downward to its next event point, say d'' on L. On the other hand, if $\#missing_color$ is zero, then $\mathcal{S}(d', z')$ is a CSS whose size can be reduced further, provided z' is not the event point of the *topmost* segment or *rightmost* segment of $\mathcal{S}(d', z')$ on $\mathcal{R}_1(d')$. In such case, we compare the two event points on $\mathcal{R}_1(d')$ which are generated by the topmost segment and rightmost segment, and then we choose that segment, say ℓ, which generates event with larger x-coordinate. If $\mathcal{C}[color(\ell)] > 1$, the segment ℓ is not *essential* and we delete it, and update the arrays $ACTIVE$, $\mathcal{C}$ and the variables *topmost*, *rightmost*. We keep on updating the topmost and rightmost variables until one of them becomes an *essential* one, i.e. $\mathcal{C}[color(\ell)] = 1$. In that case, we cannot delete any segment further and we get a *minimal-CSS* with its top-right corner defined by the event point, say e, of that segment ℓ on $\mathcal{R}_1(d')$. The size of this CSS is less than the CSS $\mathcal{S}(d, z)$. So, we update z with e and d with d', and then

shift the sweep ray $\mathcal{R}_1$ downwards to its next event. This process is continued until all the event points of $\mathcal{R}_1$ lying on L are processed and we update the *minimal-CSS*, say $\mathcal{S}_1$ accordingly.

Similarly, we move the sweep ray $\mathcal{R}_2$ upwards from its first event point to last event point on L and compute the *minimal-CSS*, say $\mathcal{S}_2$. Finally, from the two squares $\mathcal{S}_1$ and $\mathcal{S}_2$, we choose the one with minimum size, and this one will be the *minimum-CSS* among all the squares whose left side is defined by ℓ_i. Next the sweep line L moves rightward through its event points which are the left endpoints of the line segments stored in the array $\mathcal{L}_\ell$. Since a color-spanning square covers at least m different colored segments inside it, we follow the aforesaid procedures only for the first $(n-m)$ event points of L to compute the *minimal-CSS*s whose left sides are defined by those event points of L. All these *CSS* generated by the aforesaid method are of types (i), (ii) and (iii), as mentioned earlier. Similarly, to generate *minimum-CSS* of type (iv), we sweep the line L from right to left through the last $(n-m)$ sorted right endpoints of the segments that are stored in the sorted array $\mathcal{L}_r$. In these cases, the sweep ray $\mathcal{R}_1$ (resp. $\mathcal{R}_2$) is directed towards south-west (resp. north-west). The event points of $\mathcal{R}_1$ (resp. $\mathcal{R}_2$) on L along with the event points e_k of other segments on $\mathcal{R}_1$ (resp. $\mathcal{R}_2$) are generated in the similar way as that of generating the square of types (i), (ii) and (iii) and we follow the same technique to compute the *minimal-CSS* of type (iv) for $(n-m)$ event points of L. Finally, we choose the *CSS* of minimum size among all possible *minimal-CSS* of type (i), (ii), (iii) and (iv), and report it.

Theorem 4. *For a given set of n line segments $\mathcal{L}$, we can compute a minimum sized color-spanning axis-parallel square in $O(n(n-m))$ time and $O(n)$ space.*

Proof. Our algorithm computes the minimum *CSS* for all possible positions (i.e. the event points) of the sweep line L, and among all such squares, we choose the minimum one as the *minimum-CSS* for $\mathcal{L}$. The line L sweeps through $(n-m)$ event points in both forward and backward direction. For each fixed position (i.e. event point) of L, we first compute the *minimal-CSS* at the "*first event point*" of $\mathcal{R}_1$ on L in $O(n)$ time and then we compute the *minimum-CSS* on that position of L by inserting and deleting segments where each line segment in $\mathcal{L}$ is inserted at most once and deleted at most once. Hence this computation needs $O(n)$ time (amortized). To compute *minimum-CSS*, two linear sized arrays $\mathcal{C}$ and $ACTIVE$ are reused at different position of L. Hence this proves the result. □

4.2 Color-Spanning Problem Using Two Axis-Parallel Squares

Problem 4. *Given a set of n line segments $\mathcal{L} = \{\ell_1, \ell_2, \ldots, \ell_n\}$ in $\mathbb{R}^2$, where each ℓ_i is associated with one of such m distinct colors ($3 \leq m < n$). The objective is to compute two congruent CSS $\mathcal{S}_1$ and $\mathcal{S}_2$ of minimum size.*

In this problem, we compute all possible pairs of *minimal-CSS*s and choose that pair for which the size of its larger square is minimized among all other pairs.

For each segment ℓ_i, we compute all possible *minimal-CSS*s whose left sides are defined by ℓ_i. All such squares can be determined by computing *minimal-CSS* at each event point d of $\mathcal{R}_1$ (or $\mathcal{R}_2$) on L similar to process described in the Sect. 4.1 for the Problem 3. Similarly, we also compute all possible *minimal-CSS*s whose right sides (resp. top sides and bottom sides) are defined by ℓ_i. Now consider one of such *minimal-CSS*, say $\mathcal{S}_1$. For this $\mathcal{S}_1$, we must choose its corresponding square $\mathcal{S}_2$ which is *minimum-CSS* for the segments that are not covered by $\mathcal{S}_1$, and $\mathcal{S}_2$ is disjoint[2] with $\mathcal{S}_1$. One of the following statements must be true for $\mathcal{S}_2$.

(i) The square $\mathcal{S}_2$ lies to the right of the vertical line passing through the right side of $\mathcal{S}_1$.
(ii) The square $\mathcal{S}_2$ lies to the left of the vertical line passing through the left side of $\mathcal{S}_1$.
(iii) The square $\mathcal{S}_2$ lies above the horizontal line passing through the top side of $\mathcal{S}_1$.
(iv) The square $\mathcal{S}_2$ lies below the horizontal line passing through the bottom side of $\mathcal{S}_1$.

Computation of $\mathcal{S}_2$: We find the *minimum-CSS* whose left side (resp. right side, top side, bottom side) is defined by a segment ℓ_i. We do this for each $\ell_i \in \mathcal{L}$, and store each such squares in four different arrays, which corresponds to the squares defined by either the left side (resp. right side, top side, bottom side) of each segment ℓ_i. For any given square $\mathcal{S}_1$, we compute the *minimum-CSS* lying to the left (resp. right, above, below) of it in $O(n)$ time from the aforesaid four different arrays, and choose the minimum one of these four squares as $\mathcal{S}_2$. Considering each *minimal-CSS* as $\mathcal{S}_1$, we compute its corresponding square $\mathcal{S}_2$, and then return the pair having minimum size of larger square. However, if for all *minimal-CSS* $\mathcal{S}_1$, there exists no such $\mathcal{S}_2$ satisfying one of the above mentioned criteria, then no two congruent disjoint *CSS*s are possible for $\mathcal{L}$.

Theorem 5. *For a given set of n line segments in $\mathbb{R}^2$, we can compute two congruent color-spanning squares of the minimum size if such a pair exists; otherwise report no such pair exists in $O(n^3)$ time and $O(n^2)$ space.*

Proof. Since we consider all possible *minimal-CSS* $\mathcal{S}_1$ along with its corresponding minimized $\mathcal{S}_2$, our algorithm generates correct output. The computation of each *minimal-CSS* needs $O(n)$ times, and there are $O(n^2)$ such *minimal-CSS*s. For each *minimal-CSS* $\mathcal{S}_1$ we can compute its corresponding *mimimum-CSS* $\mathcal{S}_2$, in linear time. So, we need $O(n^3)$ time to determine all possible pairs ($\mathcal{S}_1$,$\mathcal{S}_2$) from which we choose the optimal pair. We need $O(n^2)$ space to store all the *minimal-CSS*. Thus the theorem follows. □

4.3 Color-Spanning Square for Simple Polygons or Disks

In this case, the inputs to the Problem 3 and Problem 4 are a set of n simple polygons or circular disks. We cover each such simple polygon (resp. circular

[2] Here disjoint means the common area of intersection of $\mathcal{S}_1$ and $\mathcal{S}_2$ is zero.

disk) exactly by an axis-parallel rectangle (resp. square). Next, replace each rectangle (resp. square) by a line segment that corresponds to any one of its diagonals. In this way, the given set of simple polygons and circular disks are mapped to the line segments. The color of these line segments are kept same as that of the given simple polygons or disks. Next, we execute the algorithm for the Problem 3 (resp. Problem 4) to compute one CSS (resp. two congruent disjoint CSS) of minimum size for the aforesaid diagonals. Now if a square $\mathcal{S}$ (resp. two congruent squares $\mathcal{S}_1$ and $\mathcal{S}_2$) covers any k such diagonals, then $\mathcal{S}$ (resp. $\mathcal{S}_1$ and $\mathcal{S}_2$) also covers their corresponding k simple polygons or circular disks, and vice versa. Hence, we obtain the following result.

Theorem 6. *For a given set of n simple polygons (resp. disks) in $\mathbb{R}^2$, we can compute (i) one minimum sized color-spanning square in $O(n(n-m))$ time and $O(n)$ space, (ii) two congruent color-spanning squares of minimum size in $O(n^3)$ time and $O(n^2)$ space.*

5 Color-Spanning Disks

Problem 5. *Given a set of n line segments $\mathcal{L} = \{\ell_1, \ell_2, \ldots, \ell_n\}$ in $\mathbb{R}^2$ and m $(3 \leq m < n)$ distinct colors, where each line segment is associated with one of such m colors. The objective is to compute one (resp. two congruent) color-spanning disk (resp. disks) of minimum radius.*

Algorithm: First, we compute a color-spanning square $\mathcal{S}$ (resp. two color-spanning squares $\mathcal{S}_1$ and $\mathcal{S}_2$) of minimum size as described in the Problem 3 (resp. Problem 4). Then, we report the circum-circle (resp. circum-circles) D of $\mathcal{S}$ (resp. D_1 and D_2 of $\mathcal{S}_1$ and $\mathcal{S}_2$ respectively) as an approximate solution of this problem.

We have the following results for the Problem 5 that follows immediately from the lower bound and approximation result of the two center problem[3] for line segments, by Sadhu et al. [8].

Lemma 1. *A lower bound for the optimal radius r^* of one (resp. two congruent disjoint) color-spanning disk (resp. disks) for a set of segments in $\mathcal{L}$ is the radius r' of in-circle of the color-spanning square $\mathcal{S}$ (resp. two congruent color-spanning squares $\mathcal{S}_1$ and $\mathcal{S}_2$) of minimum size for $\mathcal{L}$; i.e. $r' \leq r^*$.*

Theorem 7. *The algorithm stated above for the Problem 5, generates a $\sqrt{2}$ approximation result for covering the segments in $\mathcal{L}$ by one (resp. two congruent) color-spanning disk (resp. disks) of minimum radius.*

[3] In the two center problem for a given line segments $\mathcal{L}$, the objective is to cover $\mathcal{L}$ by two congruent disks of minimum (common) radius.

6 Conclusion

In this paper, we proposed a linear time algorithm to compute minimum width color-spanning vertical strip(s) for the for line segments, simple polygons and disks after $O(n \log n)$ preprocessing tasks. For these geometric objects, we also computed one (resp. two congruent disjoint) minimum sized color-spanning square (resp. squares) in $O(n(n-m))$ (resp. $O(n^3)$) time. We improved the time complexity $O(n^2 \log n)$ [1] of finding the minimum sized CSS for $\mathcal{L}$, to $O(n(n-m))$. We also showed a $\sqrt{2}$ approximation result for computing color-spanning disks of minimum radius for the segments. We leave the exact computation of two congruent color-spanning disks of minimum radius as open problem for future work.

References

1. Huttenlocher, D.P., Kedem, K., Sharir, M.: The upper envelope of Voronoi surfaces and its applications. In:7th Annual Symposium on Computational Geometry, pp. 194–203 (1991)
2. Abellanas, M., et al.: Smallest color-spanning objects. In: European Symposium on Algorithm, pp. 278–289 (2001)
3. Das, S., Goswami, P.P., Nandy, S.C.: Smallest color-spanning object revisited. Int. J.Comput. Geometry Applications, pp. 457–478 (2009)
4. Asano, T., Bhattacharya, B.K., Keil, M., Yao, F.F.: Clustering algorithms based on minimum and maximum spanning trees. In: 4th Annual Symposium on Computational Geometry, pp. 252–257 (1988)
5. Khanteimouri, P., Mohades, A., Abam, M.A., Kazemi, M.R.: Computing the smallest color-spanning axis-parallel square. In: Cai, L., Cheng, S.-W., Lam, T.-W. (eds.) ISAAC 2013. LNCS, vol. 8283, pp. 634–643. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-45030-3_59
6. Hasheminejad, J., Khanteimouri, P., Mohades, A.: Computing the Smallest Color-Spanning Equaliteral Triangle, 31st European Workshop on Computational Geometry, pp. 32–35 (2015)
7. Bae, S.W.: An algorithm for computing a minimum-width color-spanning rectangular annulus. J. Korean Inst. Inf. Sci. Eng. 246–252 (2017)
8. Sadhu, S., Roy, S., Nandy, S.C., Roy, S.: Optimal covering and hitting of line segments by two axis-parallel squares. Int. Comput. Comb. Conf. 457–468 (2017)
9. Acharyya, A., Nandy, S.C., Roy, S.: Minimum width color-spanning annulus. Theor. Comput. Sci. 16–30 (2018)
10. Pruente, J.: Minimum diameter color-spanning sets revisited. Discrete Optim. 100550 (2019)
11. Acharyya, A., Maheshwari, A., Nandy, S.C.: Color-spanning Localized query. Theor. Comput. Sci. 85–101 (2021)