Scientia Iranica D (2017) 24(3), 1325–1334

SCIENTIA IRANICA

Sharif University of Technology
Scientia Iranica
Transactions D: Computer Science & Engineering and Electrical Engineering
www.scientiairanica.com

Efficiently computing the smallest axis-parallel squares spanning all colors

P. Khanteimouri[a], A. Mohades[a,*], M.A. Abam[b], M.R. Kazemi[a] and S. Sedighin[c]

a. *Department of Mathematics and Computer Science, Amirkabir University of Technology, Tehran, Iran.*
b. *716 CE Building, Sharif University of Technology, Tehran, Iran.*
c. *3204 A.V. Williams Building, University of Maryland, College Park, Maryland, USA.*

Received 12 November 2015; received in revised form 22 August 2016; accepted 19 December 2016

KEYWORDS
Algorithm;
Location planning;
Dynamic data structure;
Color-spanning objects.

Abstract. For a set of colored points, a region is called *color-spanning* if it contains at least one point of each color. In this paper, we first consider the problem of maintaining *the smallest color-spanning interval* for a set of n points with k colors on the real line, such that the insertion and deletion of an arbitrary point takes $O(\log^2 n)$ the worst-case time. Then, we exploit the data structure to show that there is $O(n \log^2 n)$ time algorithm to compute *the smallest color-spanning square* for a set of n points with k colors in the plane. This is a new way to improve $O(nk \log n)$ time algorithm presented by Abellanas et al. [1] when $k = \omega(\log n)$. We also consider the problem of computing the smallest color-spanning square in a special case in which we have, at most, two points from each color. We present $O(n \log n)$ time algorithm to solve the problem which improves the result presented by Arkin et al. [2] by a factor of $\log n$.

1. Introduction

Background. Suppose that there are k different types of facilities such as banks, police offices, etc. and we are given n facilities of these types. A basic problem arising here is to find a region in which there is at least one representative from each type of facilities. This suggests the problem of computing *the smallest area/perimeter color-spanning objects*. Another motivation comes from *discrete imprecise data* [3,4]. In this context, each imprecise point is defined with a set of discrete possible locations. Then, for a given set of k imprecise points, we have a set of n points colored with k colors, where for each color, the points of that color represent an imprecise point. The basic problem on a set of imprecise points is to locate each imprecise point within its defining set in which a measure becomes optimized [5]. This is equivalent to choosing exactly k points with different colors in which a property, e.g. diameter, closest pair, bounding box, etc. gets minimized or maximized. Beside these two applications, this problem has other applications in statistical clustering, pattern recognition, and generalized range searching [6-8].

Related works. There are several works on colored points. Motivated from imprecise data, Ju et al. [4] showed some results on hardness of *the largest closest pair* and *the minimum planar spanning tree*. Fan et al. [9] also proposed $O(n^2 \log n)$ time algorithm to compute the expected area of convex hulls of the color-spanning sets.

In the view of location problems, for a given set of n colored points with k colors in the plane, one of the most studied problems is to find *the smallest color-spanning rectangle*. For the axis-parallel case,

*. *Corresponding author. Tel.: +98 21 6454-2542*
E-mail addresses: p.khanteimouri@aut.ac.ir (P. Khanteimouri); mohades@aut.ac.ir (A. Mohades); abam@sharif.edu (M.A. Abam); mr.kazemi@aut.ac.ir (M.R. Kazemi); sseddigh@umd.edu (S. Sedighin)

Abellanas et al. [1] showed that there are $\Theta((n-k)^2)$ minimal rectangles in the worst case and proposed an algorithm of $O((n-k)^2 \log^2 k)$ running time to solve the problem. The algorithm has been improved to $O(n(n-k)\log k)$ time by Das et al. [10]. For arbitrarily oriented rectangles, Das et al. [10] proposed an algorithm running in $O(n^3 \log k)$ time. For the problem of computing the smallest axis-parallel rectangle that spans specified number of points from each color, Barba et al. [11] presented an algorithm running in $O(n^2 k)$ time. The results are near efficient with respect to testing all minimal objects. A *minimal color-spanning object* contains at least one point from each color, and any sub-region of the same type does not contain all colors.

For the problem of computing *the smallest color-spanning circle*, Abellanas et al. [12] proposed an algorithm with $O(n^2\alpha(k)\log k)$ time using *Farthest Colored Voronoi Diagram (FCVD)*. The other approach mentioned by Abellanas et al. [1] is to obtain the smallest color-spanning circle and the smallest color-spanning axis-parallel square in $O(kn\log n)$ time using the upper envelope of Voronoi surfaces [13]. In case there are exactly two points of each color, Arkin et al. [2] presented $O(n\log^2 n)$ time algorithm to compute the smallest color-spanning square.

Our results. In Section 2, we first consider the problem of maintaining the smallest color-spanning interval for a dynamic set of colored points with k colors on the real line. We propose a data structure which spends $O(\log^2 n)$ update time per insertion and deletion using the structure designed by Overmars and van Leeuwen [14]. Next, in Section 3, we exploit the data structure to compute the smallest color-spanning axis-parallel square. The algorithm sweeps the points from top to bottom by two horizontal lines while the points between the lines are vertically projected. Then, we use the dynamic data structure to maintain the SCSI of the projected points when insertion or deletion occurs. The algorithm runs in $O(n\log^2 n)$ time and does not test every minimal candidates (in fact, we show that there may be $\Theta(kn)$ minimal color-spanning axis-parallel squares in the worst case). Hence, this result is an improvement to the result proposed by Abellanas et al. [1] in case $k = \omega(\log n)^6$. (We have recently realized that the algorithm given by Arkin et al. [2] can be generalized to get the same result.) In Section 4, we study the computation of the SCSS under the assumption that there are, at most, two points from each color. We propose $O(n\log n)$ time algorithm by reducing the problem to the 2-SAT problem. This improves the result [2] by a factor of $\log n$. Moreover, the reduction can be simply generalized to any fixed dimension, $d > 2$.

2. Dynamic maintenance of minimal color-spanning intervals

In this section, we concentrate on color-spanning intervals for a set of colored points with k colors on the real line. For ease of the presentation, we assume that points are in a general position, meaning that point coordinates are different. We first show some properties of color-spanning intervals for a static set of colored points on the real line. Next, we consider the problem of maintaining the smallest color-spanning interval for dynamic points in which the points are permitted to be inserted or deleted.

2.1. Minimal color-spanning intervals for static points

A minimal color-spanning interval for a set of colored points on the real line is an interval containing all colors, and any sub-interval of it does not contain all colors. As a simple observation, the endpoints of a minimal color-spanning interval have different colors and their colors are unique in the interval.

Suppose that we are given a set $\mathcal{P}$ of n colored points with k colors on the real line. It is easy to show that the number of minimal color-spanning intervals is linear, and they can be found with a simple algorithm in linear time and space apart from sorting. The algorithm sweeps the points from left to right with two sweep lines which stop at the endpoints of an interval. It uses an array for keeping the number of points from each color and a variable for the number of different colors between the two sweep lines. Since the sweep lines never go back, the algorithm takes $O(n)$ time and space. We omit the details due to the simplicity and conclude the following theorem.

Theorem 1. *For a given set of n points with k colors on the real line, the smallest color-spanning interval can be computed in $O(n)$ time and space apart from sorting.*

In the following, we show a new view of minimal color-spanning intervals. Gupta et al. [6] used a transformation to perform generalized range reporting/counting for a given set of colored points on the real line. Indeed, they map the original given points on the real line to points in the plane and perform the standard 3-sided range reporting/counting in the plane. We exploit the same transformation to give a new view of minimal color-spanning intervals. Let $\mathcal{P} = \{p_1, p_2, \cdots, p_n\}$ be a given set of colored points with k colors on the real line. For each color c, we first sort the points with color c (c-colored points, for short) in an increasing order. Then, for arbitrary point p_i with color c, let $pred(p_i)$ be the previous point of p_i in the list of the ordered c-colored points. In addition, we set $pred(p_j) = -\infty$ if p_j is the

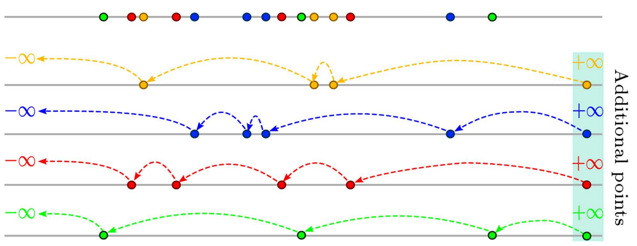


Figure 1. The ordered lists of c-colored points for each color c.

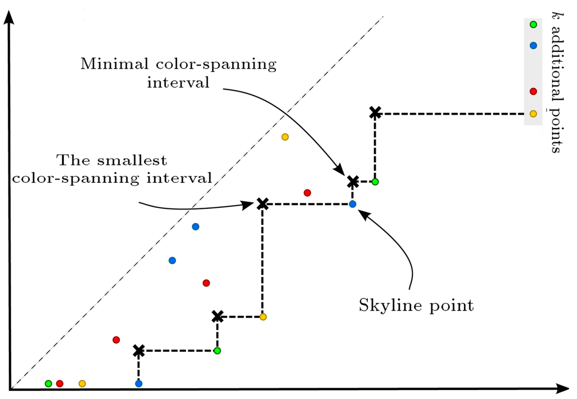

Minimal color-spanning interval
The smallest color-spanning interval
Skyline point
k additional points

Figure 2. The smallest color-spanning interval on the staircase of maximal points.

leftmost point with color c. Moreover, we insert k additional points, $\{p_{n+1}, p_{n+2}, \cdots, p_{n+k}\}$, from each color at ∞-(see Figure 1). A point, p_i, is mapped to the point $p^*_i = (p_i, pred(p_i))$ in the plane. Let $\mathcal{P}^* = \{p^*_1, p^*_2, \cdots, p^*_{n+k}\}$ be the transformed points – Figure 2 shows the transformed points of Figure 1. Furthermore, an interval $I = [l, r]$ on the real line is mapped to point $I^* = (r, l)$. This transformation has several interesting properties.

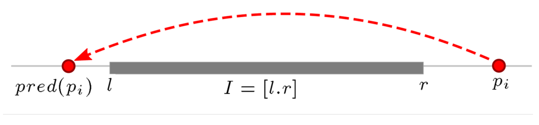
Consider an arbitrary point, $p = (p_x, p_y)$, in the plane. The vertical and horizontal lines passing through p divide the plane into four quadrants. Let $\sigma(p)$ be the right-bottom quadrant, precisely $\sigma(p) = \{q \in \mathbb{R}^2 | p_x \leq q_x, p_y \geq q_y\}$. In the following, we give some related definitions.

Definition 1. *For a set of points, $\mathcal{Q} = \{q_1, \cdots, q_n\}$, and point q in the plane, $\sigma(q)$ is $\mathcal{Q}$-empty if there is no point $q_i \in \mathcal{Q}$ in the interior of $\sigma(q)$.*

Definition 2. *For a set of points $\mathcal{Q}$ in the plane, point $q \in \mathbb{R}^2$ is maximal with respect to $\mathcal{Q}$ if $\sigma(q)$ is $\mathcal{Q}$-empty, and there is no other $\mathcal{Q}$-empty $\sigma(p)$ for some point $p \in \mathbb{R}^2$, such that $\sigma(q) \subset \sigma(p)$.*

Definition 3. *For a set of points $\mathcal{Q}$ in the plane, point $q_i \in \mathcal{Q}$ is a skyline point if $\sigma(q_i)$ is $\mathcal{Q}$-empty.*

In order to see how the maximal points in the plane are related to the minimal color-spanning intervals, we present the following lemma.



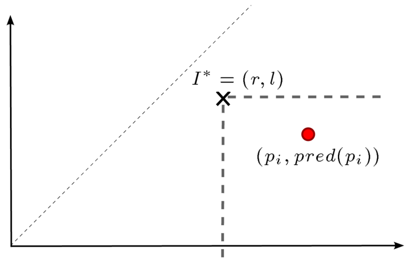


Figure 3. A not color-spanning interval $I = [l, r]$ where $\sigma(I^*)$ is not empty.

Lemma 1. *For a given set $\mathcal{P}$ of colored points on the real line, $I = [l, r]$ is a minimal color-spanning interval if and only if the point $I^* = (r, l)$ is maximal with respect to points $\mathcal{P}^*$ on the plane.*

Proof. First, suppose that $I = [l, r]$ is a minimal color-spanning interval. For the sake of contradiction, assume that $I^* = (r, l)$ is not maximal. We distinguish two cases:

1. $\sigma(I^*)$ is not empty;
2. $\sigma(I^*)$ is empty but it is not maximal.

In case (1), assume that point $(p_i, pred(p_i))$ is inside $\sigma(I^*)$ where both p_i and $pred(p_i)$ have color c - (see Figure 3). This gives us $l > pred(p_i)$ and $r < p_i$, which means $[l, r]$ is a proper subinterval of $[pred(p_i), p_i]$. Therefore, $[l, r]$ does not contain any point of color c which is a contradiction. Now, consider case (2). In this case, there is a point $q^* = (r', l')$, such that $\sigma(I^*) \subset \sigma(q^*)$. This means that there is a smaller color-spanning interval, $q = [l', r']$, contained in $I = [l, r]$, which is a contradiction. Moreover, additional points at infinity make any minimal color-spanning interval covered by maximal points on the plane. The converse implication can be proved in a similar way.

To summarize, a minimal color-spanning interval I is a maximal point on the plane with respect to $\mathcal{P}^*$. In fact, vertical and horizontal rays extending to $+y$ and $-x$ directions starting at skyline points define the maximal points (see Figure 2 for more illustration). Consider I^* as a maximal point between two consecutive skyline points $p^*_{c_i} = (p_{c_i}, p_{c_{i-1}})$ and $p^*_{c'_j} = (p_{c'_j}, p_{c'_{j-1}})$ on the plane. The minimal color-spanning interval defined by I^* is $I = [p_{c'_{j-1}}, p_{c_i}]$. Furthermore, the length of an interval is the vertical distance of its transformed point to the line $y = x$. The smallest color-spanning interval is *the minimum maximal point* on the plane (see Figure 2).

2.2. Minimal color-spanning intervals for dynamic points

In this section, we assume that the colored points on the real line are dynamic, i.e. they can be inserted or deleted. Our goal here is to maintain the minimal color-spanning intervals, specifically the smallest color-spanning interval.

In the previous section, we showed that the minimal color-spanning intervals for a set of colored points on the real line can be considered as the maximal points on the plane. Therefore, maintaining the minimal color-spanning intervals reduces to maintaining the skyline points on the plane. Overmars and van Leeuwen [14] proposed a data structure for maintaining the skyline. In the following, we briefly go over their data structure.

Overmars and van Leeuwen [14] used a binary search tree, T, which stores all points in its leaves in the sorted order by their y-coordinates. Moreover, an internal node, v, is augmented with a concatenable tree, e.g. 2-3 trees, which stores the skyline of points in the subtree rooted at v that is not contained in the skyline of points in the subtree rooted at v's parent. The skyline of all points in Q is augmented in the root of T. In addition, the cut point is where the skyline split is stored in v. If point p is inserted, a procedure $Down(T, p)$ goes downwards the tree to locate p and construct the skyline of points for subtrees of children of each internal node u on the path. Suppose that $Down(T, p)$ is running, and the skyline of u's subtree has been computed in u's parent in the previous step. Then, from the cut point of u's children, it is possible to split u's skyline at that point and to merge split parts with the lists stored in augmented trees of u's children. After inserting point p to a leaf of tree T, the other procedure $Up(T, p)$ goes up the tree to the root of T and reconstructs the augmented trees and the cut points for children of each internal node placed on the path. If a point is deleted, the procedures work similarly. These procedures split or merge two concatenable queues in $O(\log n)$ time in each internal node on the path from root to inserted or deleted leaf, and they totally need $O(\log^2 n)$ time. In addition, they showed that their data structure and algorithms use linear space [14].

Now, we exploit the described structure to maintain the smallest color-spanning interval which is the maximal point with the minimum distance to the line $y = x$ (the minimum maximal point) on the plane (see Figure 2). Let T be the same augmented binary search tree in Overmars and van Leeuwen's data structure. We modify the augmented 2-3 trees as follows. For each internal node v in augmented trees, we set two pointers, p^*_{m1} and p^*_{m2}, to the consecutive leaves which denote the minimum maximal point in the subtree rooted at v. In fact, p^*_{m1} and p^*_{m2} indicate the endpoints of the smallest color-spanning interval for the points in the subtree rooted in v. In addition, let p^*_l and p^*_r be pointers, respectively, to the leftmost and the rightmost leaves in the subtree rooted in v.

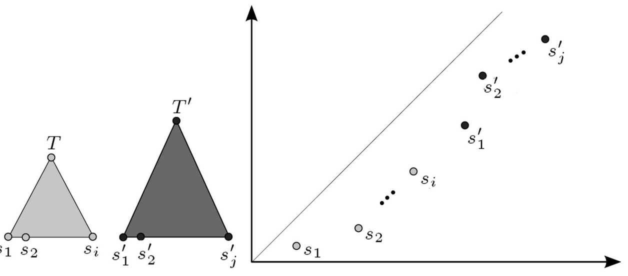


Figure 4. Two sets of skyline points, one of which is dominated by the other.

Now, consider two cases of 2-3 trees of skyline points T and T' with heights h and h' $(h < h')$ in which all points in T are dominated by any point in T'; $\forall(s \in T, s' \in T') : s_x < s'_x, s_y < s'_y$ (see Figure 4). We have the following lemma.

Lemma 2. *Merging skylines T and T' with the above condition and maintaining the minimum maximal point takes $O(h' - h)$ time.*

Proof. Assume that without loss of generality, $h < h'$. Let v be the parent of the node with height $h' - h$ on the leftmost path in T (see Figure 5(a)). We insert T into the left child of v. The node v can be either 2-node or 3-node. If v is 2-node (see Figure 5(b)) it becomes 3-node, and for updating the pointers of v, we compare the minimum maximal point of T and minimum maximal point of the subtree rooted in v.

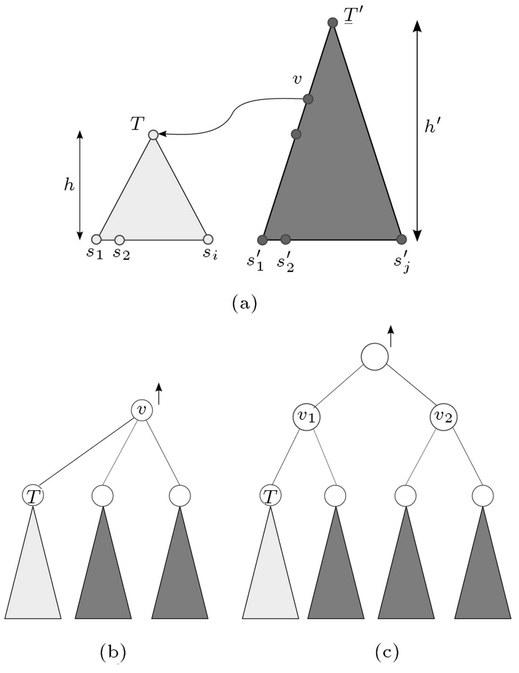


Figure 5. Cases of node v during the merge.

Moreover, the new maximal point which is defined by the rightmost point of T, and the leftmost point of $leftchild(v)$ should be considered. It is easy to update p_l^* of node v ($p^*l(v) = p_l^*(T)$). If v is 3-node (see Figure 5(c)) v should be split. It is easy to define the pointers for the newly-defined internal nodes v_1 and v_2 in constant time. After updating the pointers of v, we go up to the root of T', and for each internal node on the path to the root, we update the pointers similarly. Therefore, this procedure takes $O(h' - h)$ time.

Now, let T be 2-3 trees of skyline points. We show how we can split T into two 2-3 trees T' and T'' of skyline points with respect to some vertical line and maintain the minimum maximum point for trees T' and T''.

Lemma 3. *Splitting a skyline T into skylines T' and T'' with respect to some vertical line and maintaining the minimum maximal points in T' and T'' takes $O(\log n)$ time.*

Proof. Consider the path going down the tree with respect to x-coordinate of the vertical line (see Figure 6). It is clear that T' can be obtained by merging all subtrees T_i' on the left side of the path (similarly, T'' by merging the right subtrees T_i''). By Lemma 2, we can update the pointers to maintain the minimum maximal points for each of T' and T''. Therefore, the total computation time is $O(\log n)$ as splitting 2-3 trees.

In fact, the procedures $Down(T, P)$ and UP(T, P) in Overmars and van Leeuwen's data structure perform merge and split in the children (and their augmented trees) of each node on the path to the inserted/deleted point. Therefore, procedures $Down(T, P)$ and UP(T, P) in modified conditions still take $O(\log^2 n)$ time.

Theorem 2. *For a given set of n points with k colors on the real line, the smallest color-spanning interval can be maintained in $O(\log^2 n)$ update time per insertion and deletion and $O(n)$ space.*

Proof. When a new point p with color c is inserted

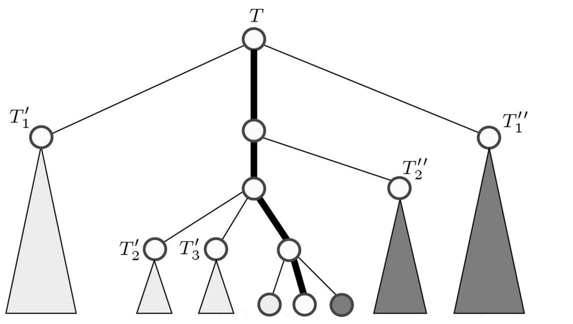


Figure 6. Splitting of a skyline T into skylines T' and T''.

between points $p_{c_{i-1}}$ and p_{c_i} in the c-colored list, point $p_{c_i}^* = (p_{c_i}, p_{c_{i-1}})$ should be deleted from the plane, and two new points $p_{c_i}^* = (p_{c_i}, p)$ and $p^* = (p, p_{c_{i-1}})$ are inserted. Deletion of a point similarly needs a constant number of insertions and deletions.

We showed how we can maintain the smallest color-spanning interval for dynamic points on the real line. This is an important tool which helps us in the next section to compute the smallest color-spanning square for a given set of colored points in the plane.

3. The smallest color-spanning square

In this section, we focus on computing the smallest area/perimeter color-spanning axis-parallel square. Suppose that we are given a set of n points with k colors on the plane. The smallest color-spanning square is an axis-parallel square which contains all colors, and its area/perimeter is minimum.

Recall that this problem can be solved using the upper envelope of Voronoi surfaces [13] in $O(kn \log n)$ time. In fact, all the previously used methods for computing the smallest color-spanning objects, such as rectangle, circle, etc. generally test all the minimal objects. We show that there are $\Omega(kn)$ minimal color-spanning squares in the worst case. This indicates that any algorithm testing all minimal color-spanning squares runs in $\Omega(kn)$ worst-case time. Next, we present an algorithm running in $O(n \log^2 n)$ time that computes the smallest color-spanning square without testing all minimal color-spanning squares.

We first explain how a minimal color-spanning square can be represented. As illustrated in Figure 7, a minimal color-spanning square is defined with two, three, or four points with different colors on its edges under the assumption that point coordinates are different (recall that this assumption is just for the ease of the presentation). In all cases, the minimal color-spanning square is bounded by two points on opposite sides. Then, it suffices to consider only the minimal color-spanning squares of case 1 instead of all. Moreover, we are interested in counting the minimal color-spanning squares with different contained points. We first give a lower bound for the number of minimal color-spanning squares.

Lemma 4. *There is a configuration of n points with k colors in the plane, in which there are $\Omega(kn)$ minimal color-spanning squares.*

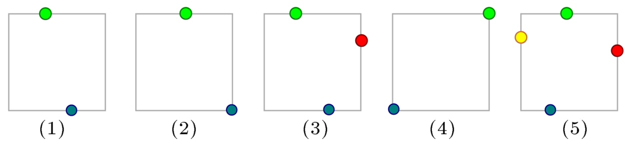


Figure 7. The cases of a minimal color-spanning square.

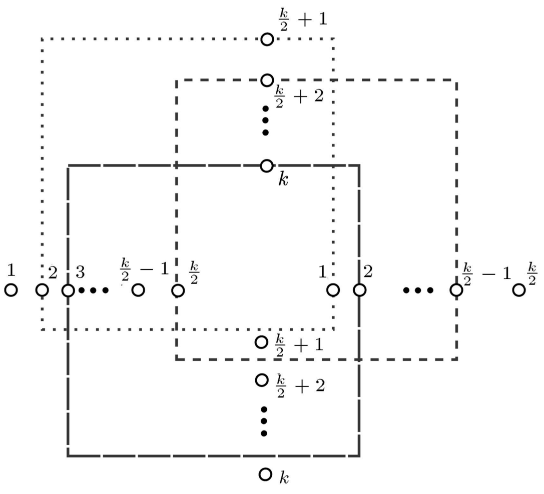


Figure 8. $\Omega(k^2)$ minimal color-spanning squares with $2k$ points.

Proof. Let's place a set of $2k$ points with k colors in the configuration, shown in Figure 8. In this pattern, we have to fix the left and the right edges in a way that they span the half of the colors (1 to $\frac{k}{2}$). So, there are $\frac{k}{2}+1$ choices for fixing the right and the left edges. Similarly, for a pair of fixed left and right edges, it is possible to obtain $\frac{k}{2}+1$ pairs to be the top and the bottom sides. Therefore, there are $\Omega(k^2)$ minimal color-spanning squares in the pattern of Figure 8 using $2k$ points. By repeating this pattern $\frac{n}{2k}$ times, we achieve $\Omega(kn)$ minimal color-spanning squares for a set of n points with k colors.

Now, we start describing the steps of our main algorithm. Suppose that the points are sorted in a descending order according to their y coordinates. The algorithm sweeps the points with two lines L_b and L_t from top to bottom, and variable d denotes the vertical distance of the sweep lines (see Figure 9 for more illustration). In fact, lines L_b and L_t, respectively, define the bottom and the top edges of the desired square. In the beginning, L_b and L_t pass through the topmost point. We move downwards the sweep lines as follows:

- Move downwards line L_t to the next point if there exists at least one color-spanning square in the strip bounded by L_b and L_t (see Figure 9(a));
- Move downwards line L_b to the next point if there is no color-spanning square in the strip bounded by L_b and L_t (see Figure 9(b)).

Now, it remains to show how we can find out if there is a minimal color-spanning square in the strip of lines L_b and L_t. Suppose that the points in the strip are projected onto the real line; let $\mathcal{P}$ be the union of this set of projected points and the additional points from each color at infinity as defined in the previous section. The following simple but important observation describes

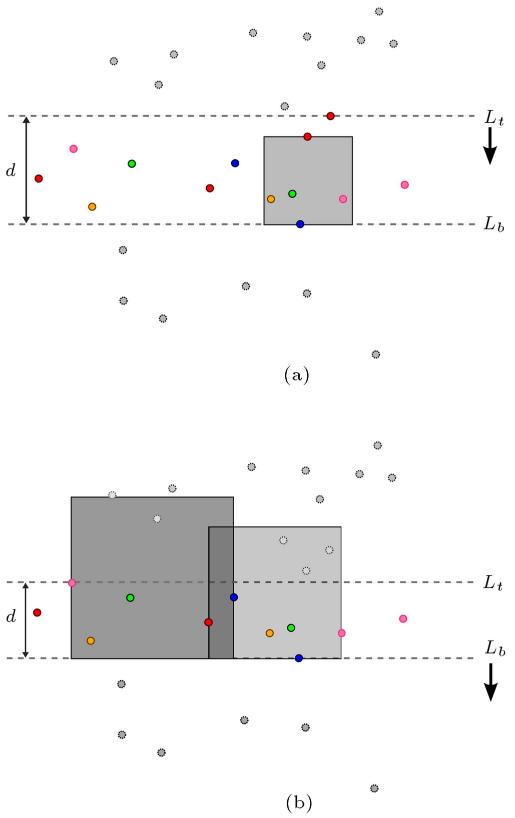


Figure 9. Sweeping the points with two lines L_b and L_t.

the necessary and sufficient conditions that a minimal color-spanning square exists inside the strip bounded by L_b and L_t.

Observation 1. *There is a minimal color-spanning square in the strip of lines L_b and L_t if and only if for the points in $\mathcal{P}$ the length of the smallest color-spanning interval is at most d.*

According to this observation, the algorithm only considers the projected points, $\mathcal{P}$. Indeed, when L_b reaches a new point or a point goes out of the strip when L_t moves downwards, we insert or respectively delete a point from the dynamic structure of maintaining the smallest color-spanning interval, described in the previous section. In addition, while L_t is moving downwards, at the time the smallest color-spanning interval becomes greater than d, the solution should be updated. We give the following theorem.

Theorem 3. *For a given set of n points with k colors in the plane, the smallest color-spanning axis-parallel square can be computed in $O(n \log^2 n)$ time.*

Proof. Let s be the smallest color-spanning square in which its top and bottom sides are defined with points p_t and p_b, respectively. Suppose that t_1 is the time L_b reaches p_b, and t_2 is the time L_t stops at point p_t. There are two possibilities. The case $t_1 < t_2$

means that L_b is at point p_b, while L_t is above point p_t. Since there is at least one color-spanning square between the sweep lines, L_t moves downwards until it reaches point p_t. When L_t leaves out p_t, the length of the smallest color-spanning interval becomes greater than d, and s has been visited. Otherwise, s is not the solution which is a contradiction. Similarly, if $t_1 > t_2$, L_t stops at point p_t, while L_b moves downwards until it reaches point p_b. Since s is the solution, L_b must stop at point p_b. Therefore, we visit the smallest color-spanning square in both cases. To analyse the running time of our algorithm, we noted that each point p enters the strip and is eliminated from it once when L_b and L_t, respectively, reach it. Therefore, in total, we have $O(n)$ insertions and deletions, and by Theorem 2, each operation spends $O(\log^2 n)$ time to maintain the smallest color-spanning interval for the points in $\mathcal{P}$. So, the algorithm runs in $O(n \log^2 n)$ time.

4. The smallest color-spanning square for pairs of points

In this section, we restrict ourselves to the case in which we have only two points from each color. Thus, the variant that we are considering is as follows.

Problem statement. Suppose that we are given n pairs (p_i, q_i) of points in the plane where points p_i and q_i are colored with color i. The goal is to compute the smallest color-spanning axis-parallel square. Arkin et al. [2] proposed an algorithm with $O(n \log^2 n)$ running time. We present an algorithm not only running faster by a factor of $O(\log n)$, but also being applicable to any fixed dimension.

Overall idea. Let the square-side length be d. Then, computing the smallest color-spanning square is reduced to the decision problem "Does there exist a square with size d covering all colors". We will introduce an $O(n)$ reduction from this decision version to the 2-SAT problem which is decidable in linear time. By this result, we can perform a binary search over the side length of the square.

Fix the side length of the square to be d. In order to find a satisfying square of the side length d, we must select at least one point from each color, such that the following two constraints hold for the selected set of points:

- The x-coordinate difference of every two selected points is at most d;
- The y-coordinate difference of every two selected points is at most d.

We associate each input point, p, with a Boolean variable, v_p. Variable v_p is true if p is a selected point. As there are $2n$ points, we have $2n$ Boolean variables. To get a valid selection, for any two points p and q (p and q may have the same color) that violate one of the above constraints, we add the following clause to the instance of the 2-SAT problem in order to ensure that at most one of them gets selected:

$$(\bar{v}_p \vee \bar{v}_q).$$

Furthermore, for each input pair of points (p_i and q_i), we add the following clause in order to ensure that at least one of them gets selected:

$$(v_{p_i} \vee v_{q_i}).$$

Since the running time of the efficient 2-SAT algorithm is $O(N + M)$ where N and M are the number of variables and clauses, respectively, the above modeling gives us an $O(n^2)$-time algorithm. In the remainder, we show that the number of clauses can be reduced to $O(n)$. The main idea is to add 4 additional Boolean variables L_p, R_p, U_p, and D_p to each input point p with the following properties:

1. If L_p is true, then none of the points q with $x(q) \leq x(p)$ gets selected;
2. If R_p is true, then none of the points q with $x(q) \geq x(p)$ gets selected;
3. If D_p is true, then none of the points q with $y(q) \leq y(p)$ gets selected;
4. If U_p is true, then none of the points q with $y(q) \geq y(p)$ gets selected.

where $x(p)$ and $y(p)$ are x and y-coordinates of p. In order to force the constraints of type 1, we add the following clauses:

$$\left(\bar{L}_p \vee \bar{v}_p\right), \quad \left(\bar{L}_p \vee L_{l(p)}\right),$$

where $l(p)$ is a point whose x-coordinate rank is immediately before that of p. Note that if L_p is true for some point p, $(\bar{L}_p \vee L_{l(p)})$ forces L_q to be true for all q where $x(q) < x(p)$. And, this together with $(\bar{L}_q \vee \bar{v}_q)$ forces v_q is not selected. We add similar clauses to the constraints of types 2, 3, and 4, as well. Using the new variables, we can avoid selecting the violating points done by $O(n)$ clauses. More precisely, for every point p, we add the following 4 clauses:

$$\left(\bar{v}_p \vee L_{l(d,p)}\right), \quad \left(\bar{v}_p \vee R_{r(d,p)}\right),$$

$$\left(\bar{v}_p \vee U_{u(d,p)}\right), \quad \left(\bar{v}_p \vee D_{d(d,p)}\right),$$

where $l(d,p)$ is the rightmost point whose x-coordinate is less than $x(p) - d$, $r(d,p)$ is the leftmost point whose x-coordinate is more than $x(p) + d$, $u(d,p)$ is the bottommost point whose y-coordinate is more than $y(p) - d$, and $d(d,p)$ is the topmost point whose y-

coordinate is less than $y(p) - d$. Finally, for every $1 \leq i \leq n$, we add:

$$(v_{p_i} \vee v_{q_i}),$$

to ensure that at least one point from each color is selected. The total number of the above clauses is $O(n)$ and is simply extendible to any constant dimension $d > 2$.

Now, we perform a binary search on d to compute the smallest color-spanning square. Let $X = \{x_1, \cdots, x_n\}$ be a sorted set of n points on the real line. The set of *interdistances* is defined by $\{x_j - x_i | x_i, x_j \in X, i < j\}$. We use matrix searching to select the ith element of the interdistances. An $m \times n$ matrix is *sorted* if each row and column of it is in a non-decreasing order. Frederickson and Johnson [16] presented an algorithm to select the ith smallest element of an $m \times n$ sorted matrix in $O(m \log(\frac{2n}{m}))$ time where $m \leq n$. By implicitly defining an $n \times n$ sorted matrix M based on the interdistances of $\mathcal{P}$ in each dimension, we can select the ith interdistance in $O(n)$ time. Let:

$$M_{i,j} = \begin{cases} x_i - x_{n-j+1} : & i > n - j + 1 \\ 0 : & \text{otherwise.} \end{cases}$$

Since the reduction can be simply generalized to any fixed dimension $d > 2$ and binary search takes $O(\log n)$ steps, we conclude the following theorem.

Theorem 4. *For a set $\mathcal{P}$ of n colored points in the d-dimensional space where there are exactly two points from each color, the smallest color-spanning hypercube can be computed in $O(n \log n)$ time.*

5. The smallest color-spanning two squares

In this section, we dedicate our attention to the problem of computing *the smallest color-spanning two squares* (SCS2S, for short). Precisely, we are given n points with k colors in the plane and the goal is to compute two axis-parallel squares which together span all colors, and the larger one is as small as possible. Similar to the previous section, we first consider the decision version of the problem where we are also given a distance d and the problem is deciding if there are two squares together spanning all colors and the side length of the larger one is at most d. Then, we use a binary search on interdistances to solve the problem.

Suppose that we are given n points with k colors in the plane. Let S_1^* and S_2^* be two squares which solve the problem of SCS2S, such that S_1^* is the larger one with side length d^*. Moreover, without loss of generality, we assume that points p_t and p_b are located in the top and the bottom sides of S_1^*, respectively (see Figure 10).

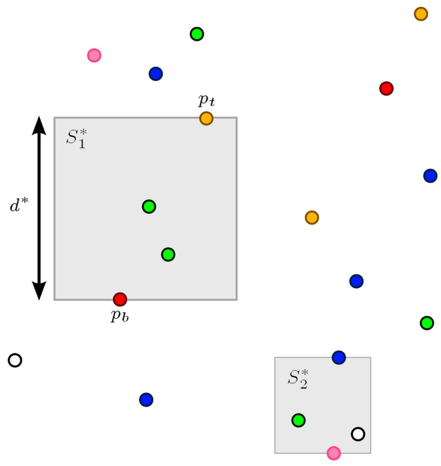


Figure 10. Two squares which solve the problem SCS2S.

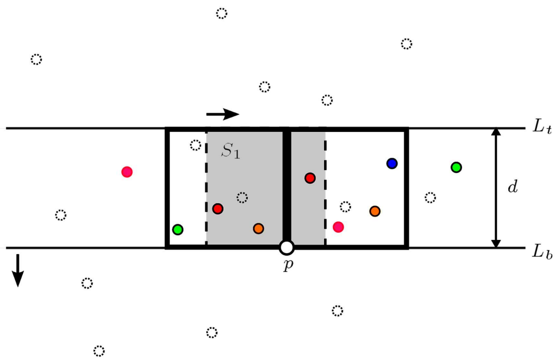


Figure 11. There are at most $2k - 2$ squares with different colors when L_b reaches point p.

In the following, for a given distance d, we show how to decide if $d \geq d^*$. In order to decide, we sweep the points with two horizontal lines L_t and L_b which their vertical distance is exactly d (see Figure 11). In fact, L_t and L_b represent the top and bottom sides of the larger square, S_1, respectively. When L_b reaches point p, we consider all the squares from left to right having p in their bottom sides. Among these squares, it suffices to consider only the squares covering different sets of colors.

Let S_1 be the leftmost square having p on its bottom side. We horizontally move S_1 to the right until its left side reaches p. During the motion $C(S_1)$, the color set covered by S_1 changes if its right side reaches the first occurrence q of some color or its left side passes the last occurrence q of some color. Otherwise, $C(S_1)$ remains unchanged. As there are at most k colors and $C(S_1)$ keeps the color of p in all squares, $C(S_1)$ may change at most $2k - 2$ times. When $C(S_1)$ changes, we create an instance of the smallest color-spanning square problem with colors not covered by S_1 and apply our algorithm in Section 3 to this instance in order to compute S_2. Whenever we find a square, S_2, whose side length is at most d, we stop searching, and we conclude that $d^* \leq d$. At the end of the sweep, if we fail finding

S_2 with the side length of at most d, we conclude that $d^* > d$.

Lemma 5. *For given n points with k colors in the plane and a distance d, we can decide in $O(kn^2 \log^2 n)$ if there are two squares together spanning all colors whose side length of the larger one is at most d.*

Proof. Let T_c be a balanced binary search tree that stores the point of color c in the strip bounded by L_t and L_b, according to their x-coordinates. We perform an insertion (deletion) into T_c when L_b (respectively L_t) reaches point p of color c in $O(\log n)$ time. Furthermore, we are able to compute the mentioned $2k-2$ points in increasing order by their x-coordinates in $O(k \log \frac{n}{k} + k \log k)$ time. Next, updating $C(S_1)$ takes $O(1)$ time while S_1 is moving to the left. To compute S_2 for the points of colors not covered by S_1, we run the algorithm described in Section 3 in $O(n \log^2 n)$ time according to Theorem 3. Therefore, for each point p, the computation takes $O(\log n + k \log \frac{n}{k} + k \log k + kn \log^2 n) = O(kn \log^2 n)$ time; in total, the running time of the algorithm is $O(kn^2 \log^2 n)$.

Now, we use a binary search algorithm in the way described in the previous section to solve the problem of SCS2S. Therefore, we present the following theorem.

Theorem 5. *For given n points with k colors in the plane, we can compute the smallest color-spanning two axis-parallel squares in $O(kn^2 \log^3 n)$ time.*

6. Conclusion

In this paper, we have presented an algorithm to maintain the smallest color-spanning interval for a set of n colored points on the real line at a cost of $O(\log^2 n)$ time for each insertion or deletion. As an application, we used this data structure to propose $O(n \log^2 n)$ time algorithm to compute the smallest color-spanning axis-parallel square for a set of n points with k colors in the plane. This improves the algorithm proposed by Abellanas et al. [1] where $k = \omega(\log n)$. However, the algorithm presented in [2] for this problem is a special case where there are only two points from each color which can be simply applied to the general case in the same running time. Nevertheless, we independently considered the problem in this special case. We modeled the problem with 2-SAT and presented $O(n \log n)$ time algorithm to solve the problem which improves the result presented in [2] by a factor of $\log n$ and can be generalized to any fixed dimension $d > 2$. Finally, we considered the problem of computing two axis-parallel squares, which altogether includes all colors, and the larger one is as small as possible. We proposed an algorithm running in $O(kn^2 \log^3 n)$ time using a binary search on the area of the larger one.

References

1. Abellanas, M., Hurtado, F., Icking, C., Klein, R., Langetepe, E., Ma, L., Palop, B. and Sacristán, V. "Smallest color-spanning objects", In *Proc. 9th Annu. European Sympos. Algorithms (ESA-2001)*, Springer-Verlag (2001).
2. Arkin, E.M., Díaz-Báñez, J.M., Hurtado, F., Kumar, P., Mitchell, J.S., Palop, B., Pérez-Lantero, P., Saumell, M. and Silveira, R.I. "Bichromatic 2-center of pairs of points", *Computational Geometry*, **48**(2), pp. 94-107 (2015).
3. Díaz-Báñez, J.M., Korman, M., Pérez-Lantero, P., Pilz, A., Seara, C. and Silveira, R.I. "New results on stabbing segments with a polygon", In *International Conference on Algorithms and Complexity*, pp. 146-157, Springer (2013).
4. Fan, C., Ju, W., Luo, J. and Zhu, B. "On some geometric problems of color-spanning sets", In *FAW-AAIM*, pp. 113-124 (2011).
5. Löffler, M. "Data imprecision in computational geometry", PhD Thesis, Utrecht University (2009).
6. Gupta, P., Janardan, R. and Smid, M. "Further results on generalized intersection searching problems: counting, reporting, and dynamization", *Journal of Algorithms*, **19**(2), pp. 282-317 (1995).
7. Matoušek, J. "On enclosing k points by a circle", *Information Processing Letters*, **53**(4), pp. 217-221 (1995).
8. Smid, M. "Finding k points with a smallest enclosing square", Report MPI-I-92-152, Max-Planck-Institut Inform., Saarbrücken, Germany (1993).
9. Fan, C., Luo, J., Zhong, F. and Zhu, B. "Expected computations on color spanning sets", In *Frontiers in Algorithmics and Algorithmic Aspects in Information and Management*, pp. 130-141, Springer (2013).
10. Das, S., Goswami, P.P. and Nandy, S.C. "Smallest color-spanning object revisited", *Int. J. Comput. Geometry Appl.*, **19**, pp. 457-478 (October, 2009).
11. Barba, L. Durocher, S., Fraser, R., Hurtado Díaz, F.A., Mehrabi, S., Mondal, D., Morrison, J.M., Skala, M. and Wahid, M.A. "On k-enclosing objects in a coloured point set", In *Proceedings of the 25th Canadian Conference on Computational Geometry*, pp. 229-234 (2014).
12. Abellanas, M., Hurtado, F., Icking, C., Klein, R., Langetepe, E., Ma, L., Palop, B. and Sacristán, V. "The farthest color voronoi diagram and related problems", *Tech. Rep.*, University of Bonn (2006).
13. Huttenlocher, D.P., Kedem, K. and Sharir, M. "The upper envelope of voronoi surfaces and its applications", *Discrete & Computational Geometry*, **9**(3), pp. 267-291 (1993).
14. Overmars, M.H. and Van Leeuwen, J. "Maintenance of configurations in the plane", *Journal of computer and System Sciences*, **23**(2), pp. 166-204 (1981).

15. Alt, H. and Scharf, L. "Computing the depth of an arrangement of axis-aligned rectangles in parallel", In *Abstracts 26th European Workshop on Computational Geometry*, pp. 33-36 (2010).

16. Frederickson, G.N. and Johnson, D.B. "Generalized selection and ranking: sorted matrices", *SIAM Journal on Computing*, **13**(1), pp. 14-30 (1984).

Biographies

Payam Khanteimouri received his MS degree in Computer Science from Sharif University of Technology in 2010. He is a PhD student in Amirkabir University of Technology, and his current research interests include computational geometry, robotics, and categorical data.

Ali Mohades is a Manager of the Computer Science Group in the Faculty of Mathematics and Computer Science in Amirkabir University of Technology in Tehran, Iran. He is the Dean of the Laboratory of Algorithms and Computational Geometry in the Faculty of Mathematics and Computer Science. His main fields of interest are computational geometry, motion planning, and facility location.

Mohammad Ali Abam is an Assistant Professor in the Department of Computer Engineering in Sharif University of Technology in Tehran, Iran, and has been in the Sharif faculty since 2011. He received his BS and MS degrees in Computer Engineering from the Sharif University of Technology, respectively, in 1999 and 2001, and PhD degree in Computer Sciences from the Eindhoven University in the Netherlands in 2007. He was a Postdoc Researcher at Aarhus university in Denmark and Dortmund University in Germany from 2008 till 2010.

Mohammad Reza Kazemi received the MS degree in Computer Science from Sharif University of Technology in 2010. He is a PhD student in Amirkabir University of Technology. His research interest has been focused on algorithms, complexity of hard problems, and approximation algorithms.

Saeed Sedighin is a third-year PhD student in the University of Maryland working under the supervision of Mohammad Taghi Hajiaghayi. His research has been mainly focused on algorithmic game theory and online algorithm design. He received his master degree from Sharif University in Computer Science.