Turkish Journal of Engineering

https://dergipark.org.tr/en/pub/tuje

e-ISSN 2587-1366

Geometric objects covering all red points and minimum blue points

Sukanya Maji*[1], Sanjib Sadhu [2]

[1]*National Institute of Technology Durgapur, Computer Science and Engineering, India, sukanyamajinitdgp@gmail.com*
[2]*National Institute of Technology Durgapur, Computer Science and Engineering, India, ssadhu.cse@nitdgp.ac.in*

Cite this study: Maji, S., & Sadhu, S. (2025). Geometric objects covering all red points and minimum blue points. Turkish Journal of Engineering, 9 (1), 47-55.

https://doi.org/10.31127/tuje.1481901

Keywords
Bichromatic point sets
Separator of point sets
Geometric Covering
Line sweeping
Algorithm

Research Article

Received:10,05,2024
Revised:07.08.2024
Accepted:21.08.2024
Published:20.01.2025

Abstract

Inspired by the applications in machine learning, we study a variation of the separation problem for a given set of bichromatic points- blue (B) and red (R) with $|B| = m$ and $|R| = n$, where these sets are separated by a geometric object. The objective of our work is to compute one or two geometric covering objects whose union covers every red point and as few blue points as possible. We consider rectangles, squares and convex polygons as the geometric covering object for the bichromatic point set. We design an $O(m+n)$ time algorithm to solve the aforesaid problem using two disjoint rectangles. For the same problem, it takes $O(m)$ time to compute a square which is used as geometric covering object. We also present an algorithm for the same problem with two disjoint squares as the geometric covering objects in $O(nm)$ time. If the geometric covering objects are two disjoint convex polygons, then it takes $O(n^2(m+n)logn)$ time. The preprocessing tasks in the algorithms for each of the aforesaid problems need $O(mlogm + nlogn)$ time and all these problems need $O(m+n)$ space.

1. Introduction

We focus our research to compute geometric covering object G for two distinct classes of given point sets so that G covers each point of one class and as few points of other class as possible. This problem is a variation of the separability problem, where we are given two sets of distinct colored (blue (B) and red (R), with $|B| = m$ and $|R| = n$) points, and a set G of geometric objects to be used as a separator for these colored points lying on $\mathbb{R}^2$. In the separability problem, we find out a separator in G which partitions the entire regions so that one of the regions contain the points in R and the other region contains only the points in B. The geometric objects may be a straight line, strip, rectangle, square or circles. If it is feasible to find out such separator, then compute it for the given point set. If there exist multiple solutions, then find out the separator that minimizes some criteria, e.g., the width of the strip, the perimeter of the rectangle, the radius of the circle etc [1-2]. In image processing and machine learning, to classify the data, geometric separability problem plays a vital role. This problem find application in the surgery of tumours in patients [3-4].

Motivation: Separability finds applications to reconstruct the urban scene by reconstructing buildings with LIDAR data. First, the data points are clustered and then the points from each of such clusters are projected onto an appropriate plane to build the structures, e.g., roofs or walls that corresponds to clusters [5-7]. Generally, the buildings' structures consist primarily of rectangles or other rectilinear forms, necessitating the identification of an appropriate rectangular contour encompassing the points. Occasionally, some points are recognized to be outside the facet under reconstruction. Consequently, the objective is to locate a shape that encompasses the facet's points (considered +ve samples) while disregarding points that are recognized to lie beyond the facet (considered -ve samples) [8-9]. We assign distinct colors to the positive and negative samples which motivates to consider rectangular objects while working with the problem of separability. Van Kreveld et al. [5-6] used rectangle as a separator G to study the problem of separability in $\mathbb{R}^2$ and proposed an algorithm with $O(nlogn)$ time.

The oncologists deal with both healthy cells and cancer cells of tumour present in the patient's body. While performing the radiation therapy or surgery to the cancer patient, the objective of the doctor is to eliminate the cancer cells as much as possible keeping intact the healthy cells. We assign red color to the cancer cells and blue color to the healthy cells to distinguish between the two types of cells. Due to such constraint of the treatment procedure, different geometric combinatorial optimization problems arise, e.g., computing the square or circle with minimum area enclosing the red points or separating blue points and red points. This inspires us to study the covering problem for the bichromatic point set by rectangles, squares which cover every red point and as few blue points as possible.

Related Works: There has been considerable research on various types of separators, both in d-dimensional (d≥2) space [10-12]. Megiddo [13] demonstrated that determining whether a line can separate two point sets, can be achieved in linear time. O'Rourke [14] focused on a different form of separators, specifically a circle C, presenting an $O(m+n)$ time algorithm to ascertain whether C can separate the two point sets. Edelsbrunner and Preparata [15] designed an algorithm to check whether a convex polygon with the fewest edges can separates two point sets, if such a polygon exists, in $O(n\ log\ n)$ time. Fekete [16] showed whether a simple polygon with fewest number of edges can separate the bichromatic point sets is NP-hard, while its approximation algorithm was designed by Mitchell [17]. Seara [18] offers an in-depth investigation into the separability issues concerning separators shaped as strips and wedges, as highlighted in previous research by Hurtado et al. [19].

Barbay et al. [20] presented a quadratic time algorithm for the Maximum-Weight Box problem that computes an axis-parallel box D with maximum weight $W(D)$ for a given weighted point set P ($|P| = n$ and the weight of each point $p \in P$ is either +ve or -ve) where $W(D)$ is given by the sum of the weights of the points lying inside D. Dobkin et al. [21] studied the Maximum Bichromatic Discrepancy Box problem, where a box that maximizes $|B - R|$, B and R being the number of blue and red points inside the box is computed. This problem was solved in $O(n^2 logn)$ time, n being the total number of blue and red points. Eckstein et al. [22] has proved that the Maximum Box problem which computes a box containing the maximum number of blue points without any red points, is NP-hard provided the dimension d is also an input to the problem. This problem for $d = 2$, was solved in $O((m+n)^2 \log(m+n))$ time by Liu and Nediak [23], and later it was improved by Backer et al. [24] which needs $O((m+n)\log^3(m+n))$ time. Bereg et al. [25] studied the maximum weighted circle (sum of weights of the points inside the circle) with minimum radius in $O(m^2(m+n)\log(m+n))$ time and $O(m+n)$ space. Abidha and Ashok [26] investigated the problem of geometric separability involving a given bichromatic point set $P = B \cup R$ (with $|P| = n$) consisting of blue (B) and red (R) points. They computed (i) fixed oriented non-uniform and uniform annulus of rectangular shape in $O(n)$ and $O(nlogn)$ time, (ii) arbitrary oriented non-uniform annulus of rectangular shape in $O(n^2 logn)$ time, (iii) fixed oriented annulus of squared shape in $O(nlogn)$ time. In the geometric variant of the red-blue set cover problem, a set of objects and a set of bichromatic points are given, the objective is to select a subset of objects to cover every blue point and as few red points as possible and this problem is NP-Hard [27]. In a weighted geometric set cover problem, $2D$ unit squares are taken as objects and for this problem, a PTAS was designed by Chan and Hu [28]. Madireddy et al. [29] showed the APX-Hardness results for defferent specialized red-blue set cover problems. Shanjani [10] proved the APX-hardness of the Red-Blue Geometric Set Cover Problem when the axis-aligned rectangles are considered as objects. Abidha and Ashok [26, 30] investigated the parameterized complexity for the generalized version of red-blue set cover problem. Bereg et al. [31] investigated the class cover problem concerning axis-parallel rectangles and designed a constant approximation algorithm. Bitner et al. [32] computed the largest separating circle for bichromatic point set in $O(m(m+n)log(m+n))$ time.

Our contributions: We design polynomial time algorithms to compute (i) an axis-parallel rectangle, (ii) two disjoint axis-parallel rectangles, (iii) an axis-parallel square, (iv) two disjoint axis-parallel squares and (v) two disjoint convex polygons that cover every red point and as few blue points as possible. To the best of our knowledge, there exist no works which are studied in this paper (see Table 1).

Table 1. Comparison Table for similar works for bichromatic points ($B \cup R$), with $|B| = m$ and $|R| = n$

Sl_ No.	Author Name	Geometric Separator	Time complexity	Space
1	Abidha and Ashok (2024) [26]	Arbitrary-oriented rectangular annulus	$O((m+n)^2 \log(m+n))$	$O(m+n)$
		Square annulus	$O((m+n) \log(m+n))$	$O(m+n)$
2	Bereg et al. (2015) [25]	Weighted circle	$O(m^2(m+n) \log(m+n))$	$O(m+n)$
3	Van Kreveld et al. (2012) [5]	Rectangle	$O((m+n) \log(m+n))$	$O(m+n)$
4	Bitner et al. (2010) [32]	Circle	$O(m(m+n) \log(m+n))$	$O(m+n)$
5	Dobkin et al. (1996) [21]	Discrepancy Box	$O((m+n)^2 \log(m+n))$	$O(m+n)$
Our works discussed in this paper – Preprocessing time: $O(nlogn + mlogm)$				
Sl No.		**Geometric Separator**	**Time complexity**	**Space**
1		Two axis-parallel rectangles	$O(m+n)$	$O(m+n)$
2		Two axis-parallel square	$O(nm)$	$O(m+n)$
3		Two convex polygons	$O(n^2(m+n) logn)$	$O(m+n)$

Outline of the Paper: We introduce the essential concepts, terminologies, and notations in Section 2. In

this paper, we first study a single rectangle and two disjoint rectangles as the geometric covering objects for the bichromatic point sets in Sections 3.1 and 3.2, respectively. Then we study the same problem using a single square and two disjoint squares as the geometric covering objects in Sections 3.3 and 3.4. In Section 3.5, two convex polygons are used as the geometric covering objects and then we conclude in Section 4.

2. Preliminaries and Notations

Throughout this paper, the rectangles and squares are assumed to be axis-parallel, unless otherwise stated. For a point p, its x-coordinate and y-coordinate are denoted by $x(p)$ and $y(p)$, respectively. For a rectangle R_1, its left side, right side, top side and bottom side are denoted by $LS(R_1)$, $RS(R_1)$, $TS(R_1)$ and $BS(R_1)$, respectively. A rectangle or a square is said to be defined by a point p, if p lies on any side of that rectangle or square. The size of a rectangle or a square is measured by its perimeter.

3. Geometric Covering Object

We study the covering of one class of point set while minimizing the other class of point set using axis-parallel rectangle(s), square(s) and convex polygons. These problems are discussed in the following subsections.

3.1. Axis-parallel rectangle

Problem 1. *Given a set $B = \{p_1, p_2, \dots, p_m\}$ of m blue points and a set $R = \{q_1, q_2, \dots, q_n\}$ of n red points; the objective is to compute an axis-parallel rectangle that cover every red point and as few blue points as possible.*

We take q_l, q_r, q_t and q_b to denote the leftmost, rightmost, topmost, and bottommost red-colored points, respectively.

Observation 1. *A minimum area axis-parallel rectangle* ℟ *that covers every red point and as few blue points as possible, must be defined by the four red points q_l, q_r, q_t and q_b which lie on the left, right, top and bottom sides of* ℟, *respectively (see Figure 1).*

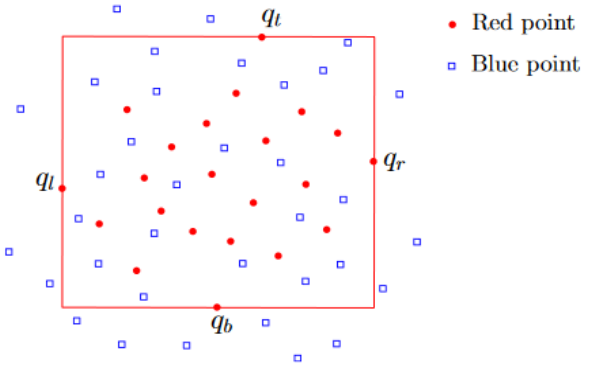


Figure 1. Rectangle ℟ that covers all the red points.

Algortihm 1

We scan all the red points in R to determine the points q_l (with minimum x-coordinate), q_r (with maximum x-coordinate), q_b (with minimum y-coordinate) and q_t (with maximum y-coordinate). We construct an axis-parallel rectangle ℟ passing through these four points q_l, q_r, q_t and q_b. The size of the rectangle ℟ cannot be reduced further keeping all the red points inside it and hence, ℟ must be the optimal solution of Problem 1 that covers all red points and as few blue points as possible. Since the four red points q_r, q_l, q_t and q_b can be obtained in $O(n)$ time, we obtain the following result.

Theorem 1. *An axis-parallel rectangle of the minimum size that covers all red points and as few blue points as possible, can be determined in $O(n)$ time by maintaining $O(1)$ extra space.*

Our solution for Problem 1 is also applicable in streaming model, where the infinite data (bichromatic points) arrives and we are restricted to use constant space to store the incoming data. In this model, we cannot store all the points due to non-availability of enough memory spaces. However, we can read the x-coordinate and y-coordinate of each point once as they arrive, and maintain four distinguished points q_l, q_r, q_b and q_t with minimum x-, maximum x-, minimum y- and maximum y-coordinates arrived so far, respectively. Thus we can report the smallest rectangle that covers all the red points and as few blue points as possible at any instant of time for Problem 1 in the streaming data model.

3.2. Two disjoint axis-parallel rectangles

Problem 2. *Given a set $B = \{p_1, p_2, \dots, p_m\}$ of m blue points and a set $R = \{q_1, q_2, \dots, q_n\}$ of n red points; the objective is to compute two disjoint axis-parallel rectangles whose union covers every red point and as few blue points as possible.*

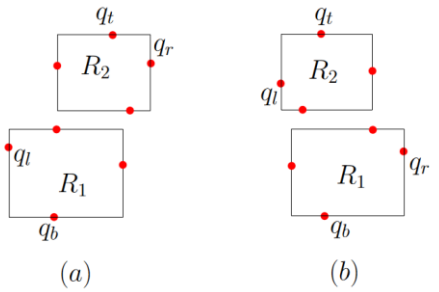
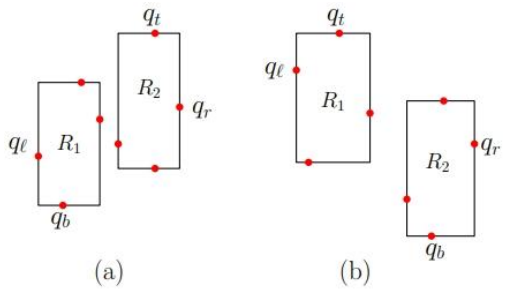


Figure 2. Disjoint rectangles R_1 and R_2 are separable by a horizontal line lying below $BS(R_2)$ and above $TS(R_1)$.

Preprocessing task: We arrange all the blue points in the increasing order of their x- (resp. y-) coordinate and store them in an array B_X (resp. B_Y). Similarly, the array R_X (resp. R_Y) stores all the red points sorted with respect to their x- (resp. y-) coordinates. This preprocessing task needs $O(nlogn + mlogm)$ time.

First, we study the properties of a pair of two disjoint axis-parallel rectangles whose union covers each red point in R. Then we compute such a pair of rectangles that cover fewest blue points. Since our objective is to find two disjoint axis-parallel rectangles whose union covers every red point, we have the following observations.

Observation 2. *Four red points q_l, q_r, q_t and q_b (defined in Section 3.1) along with another four red points in $Q \setminus$*



$\{q_l, q_r, q_t, q_b\}$ lie on the boundaries of two disjoint axis-parallel rectangles R_1 and R_2 whose union covers every red point (see Figure 2 and Figure 3).

Figure 3. Disjoint rectangles R_1 and R_2 are separable by a vertical line lying between $RS(R_1)$ and $LS(R_2)$.

Observation 3. *The two disjoint axis-parallel rectangles R_1 and R_2 must be separable either by a horizontal line (see Figure 2) or a vertical line (see Figure 3).*

Note that three of the four distinguished red points q_l, q_r, q_b and q_t (mentioned in Observation 2) may define the three sides of a rectangle as shown in Figure 4. However, all these four red points cannot lie on the boundary of one of the two rectangles for Problem 2.

Algorithm 2

We use the sweep line technique to compute all the feasible solutions of Problem 2. Without loss of generality, we take R_1 and R_2 as the lower (resp. left) and upper (resp. right) rectangles if the two rectangles R_1 and R_2 are separable by a horizontal (resp. vertical) line H (resp. V) as per Observation 3, and the two relative positions of such rectangles R_1 and R_2 are shown in Figure 2 (resp. Figure 3). Our algorithm executes in two phases to compute such pair of rectangles as follows.

Phase 1. *Computation of pair of rectangles separable by a horizontal line.*

We take a horizontal line H that sweeps upward sequentially through its event points which are the red points sorted with respect to their y-coordinates stored in R_y (see Figure 4). Without loss of generality, we assume that the sequence of red points in R_y is given by $\{q_1, q_2, \ldots, q_n\}$, i.e., $y(q_i) \leq y(q_{i+1}),\ \forall\, i \in \{1, 2, \ldots, (n\text{-}1)\}$. So, the bottommost and topmost red points q_b and q_t are also denoted by q_1 and q_n, respectively. Our algorithm always maintains two minimum-sized rectangles R_1 and R_2, so that R_1 covers all the red points lying on or below H, while R_2 covers the remaining red points lying above H. We use $\boldsymbol{h(q_i)}$ (resp. $\boldsymbol{v(q_i)}$) to denote a horizontal (resp. vertical) line passing through any red point q_i. The symbol $\boldsymbol{\ell(s)}$ is used to denote the line passing through the side s of a rectangle. The function $\boldsymbol{blue_{pt}(l_1, l_2, l_3, l_4)}$ gives a count of number of blue points lying within the rectangular region bounded by four axis-parallel lines l_1, l_2, l_3 and l_4.

Observation 4. *As the horizontal sweep line H moves in upward direction, the size of R_1 increases while that of R_2 decreases. The updated R_1 contains the previous R_1 completely inside it, whereas the previous R_2 contains the updated R_2 completely inside it.*

Observation 4 says that no blue points are removed from the previous R_1, only the blue points and a single red point are added in R_1 as the line H sweeps upwards to its next event; whereas, for R_2 some blue points and a single red point are removed without the insertion of any blue or red point into R_2. The following pseudo code shows how to generate the optimal solution of Phase 1.

Two_Axis-Parallel_Rectangles ($B \cup R$)

Input: *Bichromatic point set $B \cup R$.*

Output: *Two optimal rectangles R_1^{opt} and R_2^{opt} separable by a horizontal line, so that $R_1^{opt} \cup R_2^{opt}$ covers all red points and minimum number of blue points.*

$R_1 \leftarrow$ minimum-sized rectangle that covers q_1 and q_2.
$R_2 \leftarrow$ minimum-sized rectangle to cover the red points in $R_y \setminus \{q_1, q_2\}$.
$Count_1 \leftarrow$ number of blue points covered by R_1.
$Count_2 \leftarrow$ number of blue points covered by R_2.
$Count_{min} \leftarrow Count_1 + Count_2$;
$R_1^{opt} \leftarrow R_1$; $R_2^{opt} \leftarrow R_2$;
For $i \leftarrow 3$ to $(n-1)$ **do**
  $H \leftarrow$ horizontal sweep line passing through q_i ;
  $R_1^{old} \leftarrow R_1$; $R_2^{old} \leftarrow R_2$;
  $R_1 \leftarrow$ Updated R_1 with its top side passing through q_i and covering all the red points lying on or below H;
  $R_2 \leftarrow$ Updated R_2 with its bottom side passing through q_{i+1} and covering all the red points lying above H;
  $Count_2 \leftarrow Count_2 - blue_{pt}(\ell(BS(R_2^{old})), \ell(LS(R_2^{old})), \ell(RS(R_2^{old})), h(q_{i+1}))$;
  $Count_3 \leftarrow blue_{pt}(\ell(LS(R_1^{old})), \ell(RS(R_1^{old})), h(q_{i-1}), h(q_i))$;
  If $x(LS(R_1^{old})) \leq x(q_i) \leq x(RS(R_1^{old}))$ **then**
    $Count_1 \leftarrow Count_1 + Count_3$;
  ElseIf $x(q_i) > x(RS(R_1^{old}))$ **then**
    $Count_1 \leftarrow Count_1 + Count_3 + blue_{pt}(h(q_i), v(q_i), \ell(BS(R_1^{old})), \ell(RS(R_1^{old})))$;
  Else /* $x(q_i) < x(LS(R_1^{old}))$ */
    $Count_1 \leftarrow Count_1 + Count_3 + blue_{pt}(h(q_i), v(q_i), \ell(LS(R_1^{old})), \ell(BS(R_1^{old})))$;
  EndIf
  If $(Count_1 + Count_2 < Count_{min})$ **then**
    $Count_{min} \leftarrow Count_1 + Count_2$;
    $R_1^{opt} \leftarrow R_1$; $R_2^{opt} \leftarrow R_2$;
  EndIf
EndFor
Return R_1^{opt}, R_2^{opt};

Suppose H passes through q_{i-1} in an iteration, and then we have two rectangles R_1 and R_2 where the top side of the lower rectangle R_1 passes through q_{i-1}. We have a count of the blue points lying inside both R_1 and R_2. In next iteration, as the sweep line passes though the next red point q_i that lies immediately above q_{i-1}

(obtained from the array R_Y), we update R_1 by shifting its top side upwards to include q_i, and update R_2 by shifting its bottom side upwards to exclude q_i so that $BS(R_2)$ passes through the red point q_{i+1} lying immediately above q_i. Also note that the number of blue points in any rectangular region bounded by four lines, i.e., in $blue_{pt}()$ can be obtained from the sorted array B_X and B_Y. In this way we compute all feasible pair of rectangles where one of them lies above the other, and keep track of the pair covering fewest blue points.

Phase 2. *Computation of pair of rectangles separable by a vertical line.*

We apply the same technique analogous to Phase 1 described above to find out the pair of rectangles that are separable by a vertical line with minimum count of blue points contained by them. In this case, we need to sweep a vertical line V through its event points which are the red points stored in the array R_X (see Figure 5).

We compare the minimum count of blue points contained by the union of the pair of rectangles obtained in Phase 1 as well as Phase 2, and between them, we select the pair containing fewest blue points. Since each blue point is accessed at most twice while computing the optimal pair, the following theorem is obtained.

Theorem 2. *We can compute two disjoint rectangles whose union covers every red point and as few blue points as possible in $O(n+m)$ time and $O(n+m)$ space with a preprocessing task of $O(nlogn + mlogm)$ time.*

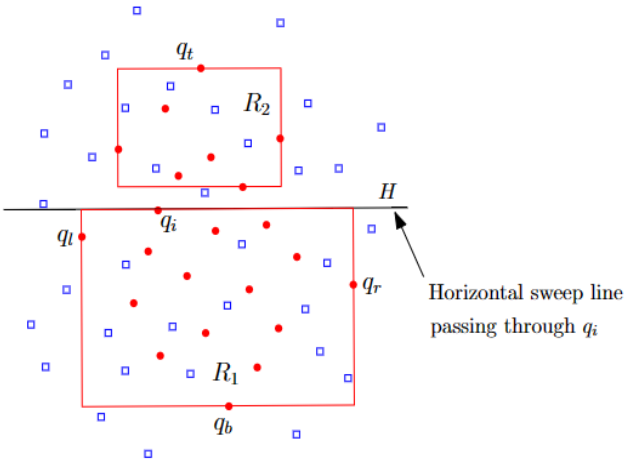


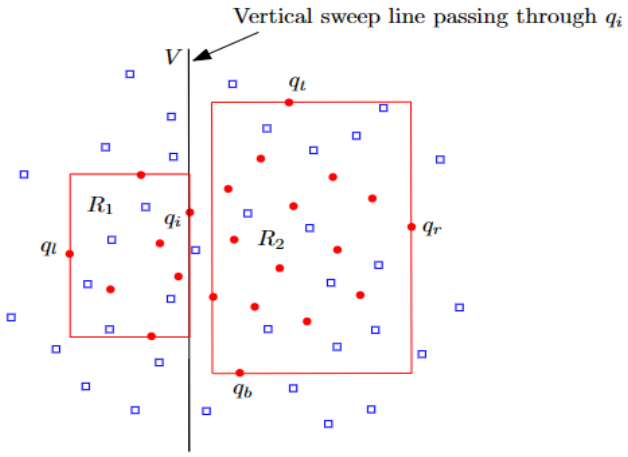
Figure 4. The red points are patritioned by the horizontal sweep line H .

3.3. An axis-parallel square

Problem 3. *Given a blue point set $B = \{p_1, p_2, \dots, p_m\}$ and a red point set $R = \{q_1, q_2, \dots, q_n\}$ with $|B| = m$, $|R| = n$; the objective is to compute an axis-parallel square that cover every red point and as few blue points as possible.*

Observation 5. *Three points uniquely define an axis-parallel square, each lying on the three different sides of the square.*

Algorithm 3



First, we compute an axis-parallel rectangle which covers all the red points as discussed in Section 3.1. The left side, bottom side, right side, and top side of this rectangle are defined by the four distinguished red points **Figure 5**. The vertical sweep line V at its event point q_i, divides the red points into two parts.

q_l, q_b, q_r and q_t, respectively. Then, we compute a minimum-sized square S that covers the aforesaid rectangle completely.

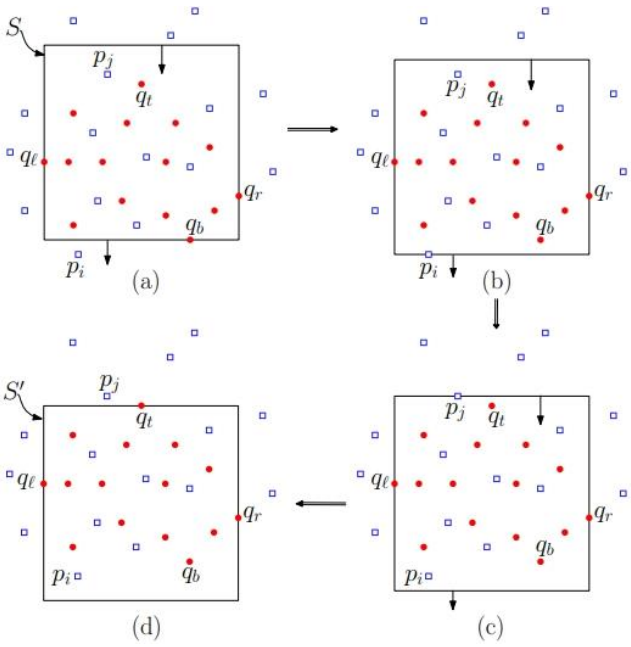


Figure 6. All possible axis-parallel squares that cover all the red points with different subset of blue points.

Note that, the length of the side of such a square is defined by the length of the longer side of the rectangle. There are three possibilities as follows.

Case 1: $|x(q_r) - x(q_l)| > |y(q_t) - y(q_b)|$

The points q_l and q_r defines the $LS(S)$ and $RS(S)$, respectively. Under this condition, $TS(S)$ (resp. $BS(S)$) may not pass through q_t (resp. q_b) in order to minimize the number of blue points covered. In Figure 6, the horizontal length is longer than the vertical length of the rectangle, and hence, the difference of $x(q_r)$ and $x(q_l)$ defines the length of the side of the square S.

Case 2: $|x(q_r) - x(q_l)| < |y(q_t) - y(q_b)|$

The points q_t and q_b define the $TS(S)$ and $BS(S)$, respectively. Under this condition, $LS(S)$ (resp. $RS(S)$) may not pass through q_l (resp. q_r) in order to minimize the number of blue points contained in S.

Case 3: $|x(q_r) - x(q_l)| = |y(q_t) - y(q_b)|$

The four points q_r, q_l, q_t and q_b define the four sides $RS(S), LS(S), TS(S)$ and $BS(S)$, respectively.

In Case 1 and Case 2, there may exist multiple squares with the (same) minimum size that covers every red point in R but different set of blue points (see Figure 6 for Case 1). However, in Case 3 only a unique square S exists. First, we discuss only the Case 1 to obtain the optimal square S^{opt} that covers all the red points and minimum number of blue points.

Using Observation 5, we have two squares S and S' (see Figure 6(a), 6(d)) for Case 1, where the bottom side of S is defined by q_b and the top side of S' is defined by q_t. We show the pseudocode of the algorithm for Case 1 as follows.

One_Axis-Parallel_Square ($B \cup R$)

Input: *Bichromatic point set $B \cup R$ satisfying Case 1.*
Output: *An optimal square S^{opt} covering all red points and minimum blue points.*

$S \leftarrow$ a square with its left, right and bottom sides passing through q_l, q_r and q_b, respectively;
$S' \leftarrow$ a square with its left, right and top sides passing through q_l, q_r and q_t, respectively;
$S^{old} \leftarrow S$;
$S^{opt} \leftarrow S$;
$Count_1 \leftarrow$ no. of blue points lying within S;
$Count_2 \leftarrow$ no. of blue points lying within S';
$Count_{min} \leftarrow Count_1$;
$B' \leftarrow$ set of blue points that lie within the rectangular region bounded by $TS(S), LS(S), RS(S)$ and $TS(S')$.
$B'' \leftarrow$ set of blue points that lie within the rectangular region bounded by $BS(S), LS(S'), RS(S')$ and $BS(S')$.
While $(B' \cup B'' \neq \phi)$ **do**
  Compute the distance d of blue point $p_i \in B'$ nearest from $TS(S)$
  Compute the distance d' of blue point $p_j \in B''$ nearest from $BS(S)$
  If $d < d'$ **then**
    shift S vertically downwards keeping its size same, so that its top side passes through p_i
    If $TS(S^{old})$ passes through a blue point **then**
      $Count_1 \leftarrow Count_1 - 1$;
      If $Count_1 < Count_{min}$ **then**
        $Count_{min} \leftarrow Count_1$;
        $S^{opt} \leftarrow S$;
      EndIf
    Else/*$BS(S^{old})$ passes through a blue or red point*/
      $Count_1$ remain same;
    EndIf
    $B' \leftarrow B' \setminus \{p_i\}$;
    $S^{old} \leftarrow S$;
  ElseIf $d > d'$ **then**
    shift S vertically downwards keeping its size same, so that its bottom side passes through p_j
    If $BS(S^{old})$ passes through a blue/red point **then**
      $Count_1 \leftarrow Count_1 + 1$;
    Else/*$TS(S^{old})$ passes through a blue point */
      $Count_1$ remain same;
    EndIf
    $B'' \leftarrow B'' \setminus \{p_j\}$
    $S^{old} \leftarrow S$;
  Else /* $d = d'$ */
    shift S vertically downwards keeping its size same, so that its top and bottom side passes through p_i and p_j, respectively.
    If $BS(S^{old})$ passes through a blue/red point **then**
      $Count_1 \leftarrow Count_1 + 1$;
    Else/*$TS(S^{old})$ passes through a blue point */
      $Count_1$ remain same;
    EndIf
    $B' \leftarrow B' \setminus \{p_i\}$;
    $B'' \leftarrow B'' \setminus \{p_j\}$;
    $S^{old} \leftarrow S$;
  EndIf
EndWhile
If $Count_2 < Count_{min}$ **then**
  $Count_{min} \leftarrow Count_2$;
  $S^{opt} \leftarrow S'$;
EndIf
If $BS(S^{opt})$ passes through a blue point **then**
  shift S^{opt} vertically upward by a very small distance $\epsilon > 0$, to remove that blue point;
  $Count_{min} \leftarrow Count_{min} - 1$;
ElseIf $TS(S^{opt})$ passes through a blue point **then**
  shift S^{opt} vertically downward by a very small distance $\epsilon > 0$, to remove that blue point;
  $Count_{min} \leftarrow Count_{min} - 1$;
EndIf
Return S^{opt};

The Case 2, i.e., if $|x(q_r) - x(q_l)| < |y(q_t) - y(q_b)|$, can be handled similarly. However, in this case, we shift the square horizontally using the same technique as described above for Case 1 to obtain the optimal one.

We access the blue points in sorted order with respect to the y-coordinate stored in the array B_Y. In each iteration, we are either adding or deleting a blue point to update the count (number of blue points covered by square), and each such blue point is added or deleted at most once. Hence, the computation of such a square needs amortized linear time. Thus, the following result is obtained.

Theorem 3. *We can compute an axis-parallel square, which covers every red point and as few blue points as possible, in $O(m)$ time and $O(n+m)$ space along with a preprocessing task of $O(nlogn + mlogm)$ time.*

3.4. Two disjoint axis-parallel squares

Problem 4. *Given a blue point set $B = \{p_1, p_2, \dots, p_m\}$ and a red point set $R = \{q_1, q_2, \dots, q_n\}$ with $|B| = m$, $|R| = n$; the objective is to compute two disjoint axis-parallel squares whose union covers every red point and as few blue points as possible.*

We represent the disjoint pair of squares by S_1 and S_2. Since the two squares are disjoint, they must be

separable either by a horizontal or a vertical line. We use two sweep lines - horizontal and vertical whose event points are all the red points stored in R_Y and R_X, respectively. For any event point $q_i \in R$ of the horizontal (resp. vertical) sweep line H (resp. V), the point set R is divided into two disjoint subsets Q_1 and Q_2, where Q_1 and Q_2 lies below (resp. to the left of) and above (resp. to the right of) the sweep line H (resp. V), respectively. Figure 7 shows an instance of the vertical sweep line V passing through its event point q_i.

Algorithm 4

First, we describe the algorithm to compute the optimal pair of disjoint squares that are separable by a vertical line. We compute two squares S_1 and S_2 that cover Q_1 and Q_2. We already know that there exist multiple squares of same size where each one covers the same set of red points (i.e., Q_1 or Q_2) but different set of blue points and among these squares, the optimal one is obtained by shifting the square and computing the count of blue points contained by the square as described in Section 3.3. For a given position of vertical sweep line V, we follow the same technique to compute two squares S_1 and S_2 that cover Q_1 and Q_2, respectively. If $RS(S_1)$ lies to the right of $LS(S_2)$ and $TS(S_1)$ lies above $BS(S_2)$, then S_1 and S_2 overlaps with each other, e.g., the squares S_1' and S_2 overlaps in Figure 7. We take an array A_1 (resp. A_2) to keep all possible squares S_1 (resp. S_2) that cover Q_1 (resp. Q_2). All the array elements, i.e., squares are ordered with respect to x-coordinate of their left sides. Note that all such squares have different set of blue points contained by them. The number of entries in each array are of the order $O(m)$ since each element (i.e., square) of the arrays, except the first and the last element, are defined by a blue point lying either on its left side or on its right side (if the squares are generated by shifting horizontally); the first and last elements (i.e., squares) of the array are defined by a red point on its right side and left side, respectively. For each square S_1 in array A_1, we can search for a square S_2 from the array A_2 so that S_2 is disjoint with S_1, and S_2 contains fewest blue points, and this step needs $O(m)$ time. We compute all pairs of squares (S_1, S_2) and choose the one for which the blue points covered by their union is minimum. This process needs $O(m^2)$ time. We repeat the above procedure for all different $(n-2)$ positions of vertical sweep line V, and hence, the total time needed is $O(nm^2)$.

Improvement: We can expedite the searching procedure by constructing another array A_{min} of the size same as that of array A_2. Let the squares stored in $A_2[i]$ be denoted by $S_2[i]$. We define a 1-dimensional array A_{min}, whose i^{th} entry $A_{min}[i]$ stores the square with minimum number of blue points among the set of squares stored in $A_2[i]$ through $A_2[k]$, where k is the rightmost index of array A_2. The $A_{min}[i]$ is defined as below

$$A_{min}[i] = \begin{cases} A_2[i], & if\ i = k, \text{the rightmost index of } A_2 \\ \text{minimum}(A_{min}[i+1], A_2[i]), & \text{otherwise} \end{cases}$$

Here, the function minimum() returns the square containing fewest number of blue points. We populate the array A_{min} in the reverse direction (i.e., from the rightmost entry to the leftmost entry) by reading the array A_2 in the backward direction from right to left. Thus we can compute all the entries of A_{min} in linear time. Now for a square S_1 in A_1, we can search for an element (i.e., a square) in A_{min}, say $A_{min}[j]$ whose left side lies to the right of $RS(S_1)$, while the left side of the square stored in preceding index, i.e., $A_{min}[j-1]$ lies to the left of $RS(S_1)$. Thus, for the square S_1, the square (lying to the right of S_1) with minimum number of blue points covered is obtained in $A_{min}[j]$. When we consider the next square S_1 lying to the right of previous S_1 in A_1, then we continue to search square in A_{min} from the position where we stopped searching in the previous iteration, since the right side of the corresponding square S_2 must not occur to the left of $A_{min}[j]$. Thus, we can determine all pairs of disjoint squares (S_1, S_2) in amortized $O(m)$ time, where the square S_2 contains minimum number of blue points for its corresponding square S_1. Among these pairs, we choose the one whose union contains fewest blue points. We repeat the above procedure at each event point of the vertical sweep line. Since the number of event points (which are the red points in set R) for the sweep line V are $O(n)$, it needs $O(nm)$ time to compute the optimal pair of squares that are separable by a vertical line.

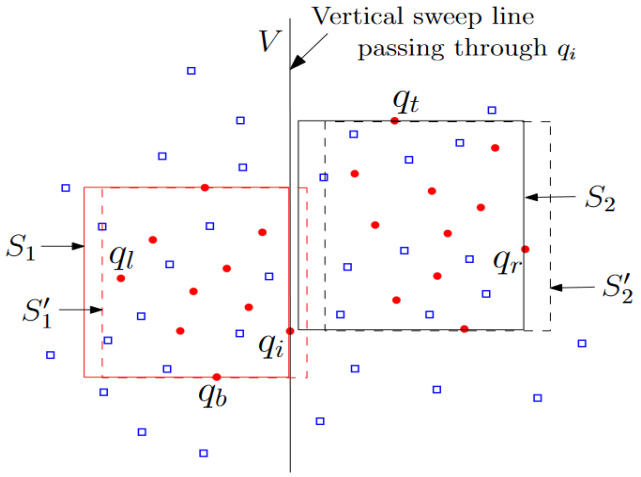


Figure 7. Pairs of disjoint squares (S_1, S_2) and (S_1', S_2') when the vertical sweep line V passes through q_i.

Similarly, we can compute the optimal pair of squares that are separable by a horizontal line by sweeping horizontal line H through the red points stored in R_Y. Finally, between the pairs of disjoint squares obtained by sweeping the vertical line V and the horizontal line H, we choose the pair of squares whose union covers fewest blue points and report that pair as the optimal solution to Problem 4. Therefore, the following result is obtained.

Theorem 4. *We can compute two disjoint axis-parallel squares whose union covers every red point and as few blue points as possible in $O(nm)$ time using $O(n+m)$ space along with $O(nlogn + mlogm)$ preprocessing time.*

3.5. Two disjoint convex polygons

Problem 5. *Given a blue point set $B = \{p_1, p_2, \dots, p_m\}$ and a red point set $R = \{q_1, q_2, \dots, q_n\}$ with $|B| = m$, $|R| = n$; the objective is to compute two disjoint convex polygons,*

whose union covers every red point and as few blue points as possible.

Algorithm 5

A minimum-sized convex polygon that covers a set of red points must be a convex hull for that point set. We use convex hull as a geometric tool to solve Problem 5 and the pseudocode of the algorithm is shown below.

Two_Convex_Polygons ($B \cup R$)

***Input**: Bichromatic point set $B \cup R$.*
***Output:** Two optimal convex polygons CH_1^{opt} and CH_2^{opt} covering all red points and minimum number of blue points.*
$Count_{min} \leftarrow \infty$; /* stores the number of blue points covered by the union of two optimal convex polygons */
For $i = 1$ to $(n-1)$ **do**
  For $j = (i+1)$ to n **do**
    $L \leftarrow$ The line passing through the pair of red points (q_i, q_j) and directed from q_i to q_j;
    $Q_\ell \leftarrow$ The set of red points lying to the left of L;
    $Q_r \leftarrow$ The set of red points lying to the right of L;
    $CH_1 \leftarrow$ Convex hull of the red points in $Q_\ell \cup \{q_i, q_j\}$;
    $Count_1 \leftarrow$ no. of blue points lying inside CH_1;
    $CH_2 \leftarrow$ Convex hull of the red points in Q_r;
    $Count_2 \leftarrow$ no. of blue points lying inside CH_2;
    If $(Count_1 + Count_2 \leq Count_{min})$ **then**
      $Count_{min} \leftarrow Count_1 + Count_2$;
      $CH_1^{opt} \leftarrow CH_1$;
      $CH_2^{opt} \leftarrow CH_2$;
    EndIf
  EndFor
EndFor
Return CH_1^{opt}, CH_2^{opt};

The number of separator lines L that passes through every pair of red points is ${}^nC_2$, i.e., $O(n^2)$. For a particular L, we compute each of the corresponding two convex hulls in $O(nlogn)$ time [10]. We can check whether a blue point lies inside a convex hull in $O(logn)$ time, and we repeat this step for all m blue points in B. Hence, we can count the number of blue points contained by the pair of convex hulls, i.e., convex polygons in $O(mlogn)$ time. Thus, for a line L, it takes $O((m+n)logn)$ time to construct the pair of convex hulls and to count the total number of blue points lying inside them. The number of such lines L are $O(n^2)$ and hence, the following result is obtained.

Theorem 5. *We can compute two disjoint convex polygons whose union covers all the red points and as few blue points as possible, in $O(n^2(m+n)logn)$ time and $O(n+m)$ space along with $O(nlogn + mlogm)$ preprocessing time.*

4. Conclusion

In this work we have shown how to compute geometric objects such as rectangles, squares, convex polygons whose union covers every red point and as few blue points as possible. In future, it remains a challenge to solve these problems in higher dimensions.

Author contributions

Sukanya Maji: Conceptualization, Methodology, Writing-Original draft preparation, Validation. **Sanjib Sadhu:** Validation, Visualization, Investigation, Writing-Reviewing and Editing.

Conflicts of interest

The authors declare no conflicts of interest.

References

1. Acı, M., Acı, Ç. İ., & Avcı, M. (2018). Performance comparison of anfis, ann, svr, cart and mlr techniques for geometry optimization of carbon nanotubes using castep. Turkish Journal of Engineering, 2(3), 119-124.
2. Dennison, R., Dasebenezer, G. K., & Dennison, R. (2024). Cervic cancer classification using quantum fuzzy set. Turkish Journal of Engineering, 8(4), 687-694
3. Aydin, O. F., & Gökaşar, İ. (2021). The effects of including social factors in ride-matching algorithms on the performance and the quality of matches. Turkish Journal of Engineering, 5(1), 41-47.
4. Juraev, D. A., & Bozorov, M. N. (2024). The role of algebra and its application in modern sciences. Engineering Applications, 3(1), 59-67.
5. Kreveld, M. V., Lankveld, T. V., & Rie, M.D. (2012). (α, δ)-Sleeves for reconstruction of rectilinear building facets. In Progress and New Trends in 3D Geoinformation Sciences. Springer Berlin Heidelberg, 231-247.
6. Kreveld, M.V., Lankveld, T.V., & Veltkamp, R. (2009). Identfying well-covered minimal bounding rectangles in 2D point data. In 25th European Workshop on Computational Geometry (EuroCG), 277-280.
7. Lankveld, T.V., Kreveld, M.V., & Veltkamp, R. (2011). Identifying rectangles in laser range data for urban scene reconstruction. Computers & Graphics, 35(3), 719-725.
8. Maji, S., Pandit, S., & Sadhu, S. (2024). Generalized Red-Blue Circular Annulus Cover Problem. CoRR abs/2402.13767.
9. Maji, S., Pandit, S., & Sadhu, S. (2023). Red-Blue Rectangular Annulus Cover Problem. In International Workshop on Frontiers in Algorithmics (IJTCS-FAW), 195-211.
10. Shanjani, S.H. (2020). Hardness of approximation for red-blue covering. In: Canadian Conference on Computational Geometry (CCCG), 39–48.
11. Preparata, F. P., & Shamos, M. I. (2012). Computational Geometry: An Introduction. Springer Science & Business Media.
12. de Berg, M., Kreveld, M.V., Overmars, M., Schwarzkopf, O. (2000). Computational Geometry: Algorithms and Applications. Springer Science & Business Media.

13. Megiddo, N. (1983). Linear-time algorithms for linear programming in R^3 and related problems. SIAM journal on computing, 12(4), 759-776.
14. O'Rourke, J., Kosaraju, S.R., & Megiddo, N. (1986). Computing circular separability. Discrete & Computational Geometry, 1, 105-113.
15. Edelsbrunner, H., & Preparata, F.P. (1988). Minimum polygonal separation. Information and Computation, 77, 218-232.
16. Fekete, S. (1992). On the complexity of min-link red-blue separation. Manuscript, department of applied mathematics, SUNY Stony Brook, NY.
17. Mitchell, J.S.B. (1993). Approximation algorithms for geometric separation problems. In Technical report. State University of New York at Stony Brook
18. Seara, C. (2002). On geometric separability. Applied Mathematics, Ph. D. Thesis, University of. Polit'ecnica de Catalunya.
19. Hurtado, F., Noy, M., Ramos, P. A., & Seara, C. (2001). Separating objects in the plane by wedges and strips. Discrete Applied Mathematics, 109(1-2), 109-138.
20. Barbay, J., Chan, T.M., Navarro, G., & P´erez-Lantero, P. (2014). Maximum-weight planar boxes in O(n^2) time (and better). Information Processing Letters. 114(8), 437–445.
21. Dobkin, D.P., Gunopulos, D., & Maass, W. (1996). Computing the maximum bichromatic discrepancy with applications to computer graphics and machine learning. Journal of Computer and System Sciences. 52(3), 453–470.
22. Eckstein, J., Hammer, P.L., Liu, Y., Nediak, M., & Simeone, B. (2002). The maximum box problem and its application to data analysis. Computational Optimization and Applications. 23(3), 285–298.
23. Liu, Y., & Nediak, M. (2003). Planar case of the maxium box and related problems. In: Proceedings of the 15th Canadian Conference on Computational Geometry (CCCG), Halifax, Canada, 14–18.
24. Backer, J., & Keil, J.M. (2010). The mono- and bichromatic empty rectangle and square problems in all dimensions. In: LATIN 2010: Theoretical Informatics, 9th Latin American Symposium (LATIN), Proceedings,14–25.
25. Bereg, S., Daescu, O., Zivanic, M., & Rozario, T. (2015). Smallest maximum-weight circle for weighted points in the plane. In: Computational Science and Its Applications (ICCSA) - 15th International Conference, Proceedings, 244–253.
26. Abidha, V.P., & Ashok, P. (2022). Geometric separability using orthogonal objects. Information Processing Letters 176, 106245.
27. Carr, R.D., Doddi, S., Konjevod, G., & Marathe, M. (1999). On the red-blue set cover problem. Technical report, Sandia National Lab (SNL-NM), Albuquerque, NM (United States), Sandia, 345-353.
28. Chan, T.M., & Hu, N. (2015). Geometric red–blue set cover for unit squares and related problems. Computational Geometry. 48(5), 380–385.
29. Madireddy, R.R., Nandy, S.C., & Pandit, S. (2021). On the geometric red-blue set cover problem. International Workshop on Algorithms and Computation (WALCOM), 129–141, Springer
30. Abidha, V., & Ashok, P. (2024). Red blue set cover problem on axis-parallel hyperplanes and other objects. Information Processing Letters. 186, 106485.
31. Bereg, S., Cabello, S., Díaz-Báñez, J.M., Pérez-Lantero, P., Seara, C., & Ventura, I. (2012). The class cover problem with boxes. Computational Geometry 45(7), 294–304.
32. Bitner, S., Cheung, Y., & Daescu, O. (2010). Minimum separating circle for bichromatic points in the plane. In 2010 International Symposium on Voronoi Diagrams in Science and Engineering (ISVD), 50-55. IEEE.