information

MDPI

Article

Extensions of the Maximum Bichromatic Separating Rectangle Problem †

Bogdan Armaselu

Independent Researcher, Flower Mound, TX 75028, USA; barmaselub@gmail.com

† This is an extended version of the CCCG'2021 conference paper with the same title.

Abstract: An important topic in the field of geometric optimization is finding the largest rectangle separating two different points sets, which has significant applications in circuit design and data science. We consider some extensions of the maximum bichromatic separating rectangle (MBSR) problem. In one of the extensions, the optimal rectangle may include up to k outliers, where k is given as part of the input. This extension and is called *MBSR with outliers* or MBSR-O. In this paper, we improve the current known time bounds for MBSR-O from $O(k^7 m \log m + n)$ to $O(k^3 m + km \log m + n)$ using a clever staircase sweep approach. We also propose another extension, which is named *MBSR among circles* or MBSR-C and asks for the largest rectangle separating red points from blue unit circles. Our solution to MBSR-C is an $O(m^2 + n)$-time algorithm that involves an optimized scanning of all candidate circle arcs for locations of potential optimal solutions.

Keywords: extensions; bichromatic; maximum; separating rectangle; outliers; circles

check for updates

Citation: Armaselu, B. Extensions of the Maximum Bichromatic Separating Rectangle Problem. *Information* **2022**, *13*, 476. https://doi.org/10.3390/info13100476

Academic Editor: Giovanni Viglietta

Received: 1 August 2022
Accepted: 28 September 2022
Published: 2 October 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

1. Introduction

Geometric optimization deals with optimization problems involving large sets of geometric objects. Bichromatic separability of point sets is a well-known topic in the field of geometric optimization. Typically, we are given a set of "red" points and a set of "blue" points in two or three dimensions, and the goal is to separate them using various geometric loci, such as lines, planes, circles, spheres, rectangles, or boxes.

The Maximum Bichromatic Separating Rectangle (MBSR) problem was introduced by Armaselu et al. in [1] (see also [2]) and is stated as follows. Given a red point set R and a blue point set B in the plane, with $|R| = n, |B| = m$, compute the axis-aligned rectangle S satisfying the following:

(1) S contains all points in R;
(2) S contains the fewest points in B among all rectangles satisfying (1);
(3) S has the largest area of all rectangles satisfying (1) and (2).

Such a rectangle is called *maximum bichromatic separating rectangle* (MBSR) or simply *largest separating rectangle*.

Denote the smallest axis-aligned rectangle enclosing R by S_{min}.

In this paper, we consider two extensions of the MBSR problem.

The first extension, introduced in [3], is called *MBSR with outliers* (MBSR-O) or simply *outliers version*. It seeks to find the largest axis-aligned rectangle containing all red points and up to k blue points outside S_{min}, where k is given as part of the input. That is, MBSR-O is a relaxation of condition (2) from the original MBSR problem. The running time of the algorithm in [3] is $O(k^7 mlogm + n)$. However, when the k is large (e.g., $k = \Theta(m)$), this running time bound can be unreasonably high. We will show how to improve this time bound to $O(k^3 m + km \log m + n)$ using a more clever sweep line-based approach.

We also introduce another extension of MBSR, called *MBSR among circles* (MBSR-C) or simply *circles version*, in which there are red points and blue unit circles, and the goal is to find the largest rectangle containing no point of any blue circle outside S_{min} while containing all red points.

For both extensions, we assume no unbounded solution and also that all points are in general positions.

The outliers version can have applications in various domains. For instance, in VLSI or circuit design, one might seek to place a hardware component (e.g., cooler) on a board with minor fabrication defects (blue points), where up to k defects are tolerated for component placement. The red points may indicate "hot spots" that must be covered by the component on the board.

The circles version is motivated by problems involving "imprecise" data, such as probabilistic applications, machine learning applications, and tumor extraction with large or imprecise cells as red or blue points. For instance, the goal is to surgically remove a tumor using a rectangular tool, with tumor cells marked by red points while blue points denote healthy cells and osteoclasts that should not be removed or cut out.

Various other applications of bichromatic separation with imprecise points can be found in spatial databases and data science.

1.1. Related Work

Geometric separability of point sets, which deals with finding a geometric locus that separates two or more point sets whilst achieving a specific optimum criterion, is an important topic in computational geometry. Various approaches deal with finding a specific type of separator (e.g., hyperplane) when the points are guaranteed to be separable. However, this is not always the case, hence the need for results on weak separability, i.e., either minimizing the misclassifications or allowing up to a fixed number of them.

The problem of finding the smallest separating circle among red and blue points (i.e., containing all red points and the fewest blue points), was introduced by Bitner and Daescu et al. [4]. They provide two algorithms that find all optimal solutions: the first one runs in $O(m^{1.5}\log^{O(1)} n + n\log n)$ time and the second one runs in $O(mn\log m + n\log n)$-time. The dynamic version of the problem, in which blue points may be dynamically inserted and deleted at run time, was later addressed by Armaselu and Daescu [5], who provided three data structures foir this version. The first one is a unified data sdtructure supporting both insertion and deletion queries in $O(n\log m)$ time, as well as $O((m+n)\log m)$ time updates. The other two are deletion-specific (resp., insertion-specific) and allow $O(\log^2(mn))$ (resp., $O(\log(mn))$) query time, at the expense of $O(mn\log(mn))$ update time.

Armaselu and Daescu were the first to address the MBSR problem. Their algorithm runs in $O(m\log m + n)$ time [1,2]. When the axis-alignment restriction on the MBSR is dropped, they have an $O(m^3 + n\log n)$ time algorithm. They also come up with an $O(m^2(m+n))$-time algorithm to compute the maximum-volume separating box in three dimensions [2]. Later, this wasr improved to $O(m^2 + n)$ time [6].

Separability of imprecise points has also been considered. In such setting, points are asscoiated with an region of imprecision. For instance, blue unit circles (i.e., MBSR-C) can be thought of as imprecision regions, where the imprecise points are their centers. When the imprecisions are axis-aligned rectangles, de Berg et al. [7] come up with a linear-time algorithm to compute certain separators, i.e., that are 100% likely to separate the point sets. They also show how to compute possible separators (>0% likely to separate) in $O(n\log n)$ time.

It is worth mentioning that all these results on separators deal with blue points. However, there are also results for blue obstacles. In a more recent paper [3], Armaselu, Daescu, Fan, and Raichel give an algorithm to find a largest rectangle separating red points from blue axis-aligned rectangles in $O(m\log m + n)$ time.

Computing the largest empty (axis-aligned) rectangle problem is a very popular and studied topic. Given a set of planar points P, the goal is to compute the largest axis-aligned rectangle that has a point $p \in P$ on each of its sides but none inside it. For the axis-aligned version, Agarwal et al. [8] provided the best currently known time bound, $O(n\log^2 n)$, for computing *one* optimal solution, while Hsu et al. [9] got the best-known result for computing all optimal solutions, namely, in $O(n\log n + M)$ time, where M is the number of maximal empty rectangles. Mukhopadhyay et al. [10] solved the arbitrary orientation

version and gave an $O(n^3)$ time algorithm that outputs all optimal solutions. In addition, Chaudhuri et al. [11] proved a lower bound of $\Omega(n^3)$ optimal solutions in the worst case.

Nandy et al. considered the problem of finding the maximal empty axis-aligned rectangle among a given set of rectangles isothetic to a given bounding rectangle [12]. They show how to solve the problem in $O(n \log n + R)$ time, where R is the number of rectangles. Later, they solved the version where obstacles have arbitrary orientation using an algorithm that takes $O(n \log^2 n)$ time [13]. Finally, they consider the problems of locating the largest empty rectangle inside a simple polygon, as well as avoiding a bunch of simple polygons [13].

1.2. Our Results

First, we improve upon the MBSR-O result in [3]. Specifically, we provide a slight improvement of $O(k^7 m + m \log m + n)$ time for k outliers, which we further improve to $O(k^3 m + km \log m + n)$ time. The latter is faster than the former whenever $k > (\log m)^{\frac{1}{4}})$. After that, we come up with an $O(m^2 + n)$ time algorithm for MBSR-C.

The rest of the paper is structured as follows. In Section 2, we describe our improvements to MBSR-O, in Section 3 we describe our solution to MBSR-C, and finally, in Section 4 we draw the conclusions and suggest some future directions.

2. Finding the Largest Separating Rectangle with k Outliers

The goal is to compute the largest axis-aligned rectangle enclosing R while containing no more than k blue points of B ("outliers"), for a given $k \geq 0$. We call this rectangle maximum bichromatic separating rectangle with outliers (MBSR-O).

We first discard the blue points inside S_{min}, as they cannot be avoided.

In [3], the given algorithm for MBSR-O operates as follows. We first compute the smallest rectangle S_{min} enclosing R in linear time, and then partition the space outside S_{min} into 8 regions, using the lines bounding S_{min}. Specifically, there are 4 "corner" regions NE, NW, SW, SE. (also known as "quadrants"), and 4 "side" regions E, N, W, S. In each such region Q, we consider the set B_Q of blue points inside Q.

Definition 1. *[3] A point $p \in B_{NE}$ dominates another point $q \in B_{NE}$, if $x(p) > x(q)$ and $y(p) > y(q)$. Similarly, a point $p \in B_{NW}$ dominates another point $q \in B_{NW}$, if $x(p) < x(q)$ and $y(p) > y(q)$, a point $p \in B_{SE}$ dominates another point $q \in B_{SE}$, if $x(p) > x(q)$ and $y(p) < y(q)$, and a point $p \in B_{SW}$ dominates another point $q \in B_{SW}$, if $x(p) < x(q)$ and $y(p) < y(q)$.*

Definition 2. *[3] For each B_Q and for any t such that $0 \leq t \leq k$, the t-th level staircase of B_Q is the rectilinear polygon formed by the blue points in B_Q that dominate exactly t blue points in B_Q.*

Note that an optimal solution contains t points from B_Q if and only if it is bounded by the t-th level staircase of B_Q, shown in Figure 1.

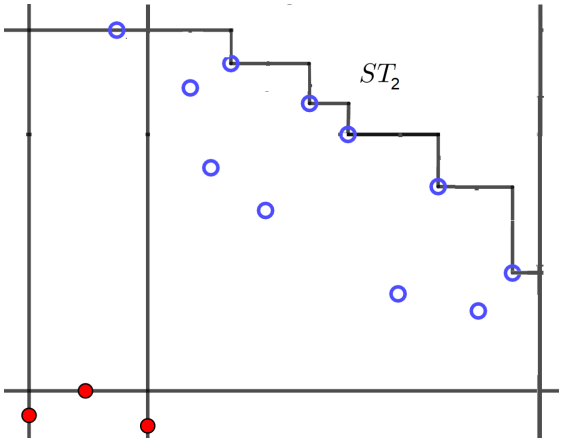

Figure 1. The 2nd level staircase of B_{NE}.

For each partition of k into 8 smaller natural numbers $k = k_E + k_{NE} + \cdots + k_{SE}$, that is, one natural number for each region, we do the following.

1. Consider the $k_Q + 1$-th closest to S_{min} blue point from each side region Q. Note that, by extending S_{min} in each direction until reaching these points, one obtains a rectangle S_{max} which definitely contains the target rectangle.
2. For each quadrant Q, compute the k_Q-level staircase $ST_{k_Q}(Q)$ of Q in $O(m \log m)$ time.
3. Solve a "staircase" problem on S_{min}, S_{max}, and ST in $O(m)$ time. That is, compute the largest rectangle enclosing S_{min}, supported by points of the staircases, and contained in S_{max}.

Since there are $O(k^7)$ ways of partitioning the integer k into 8 smaller natural numbers, it follows that the running time of this approach is $O(k^7 m \log m + n)$.

2.1. A Slight Improvement on the Running Time

We improve the running time bounds for MBSR-O. To do that, instead of computing the t-th level staircase $ST_t(Q)$ for each quadrant and each natural number $t \leq k$, we precompute all staircases in a more clever fashion that involves sweeping the blue points in each quadrant Q with a line and computing all $ST_t(Q)$'s in a single sweep.

Lemma 1. *The t-level staircases $ST_t(Q)$ can be computed in $O(m \log m + mk)$ time for all quadrants Q and integers $t \leq k$.*

Proof. We show how to compute $ST_t(NE)$ for all $t \leq k$ with a sweep line algorithm, as for other quadrants the approach is similar. For simplicity, denote $ST_t(NE)$ as ST_t, that is, without specifying the quadrant. Sort and label the points in B_{NE} by increasing x-coordinate, and denote the resulting sequence as $p_1, \ldots, p_m$. Sweep a vertical line l from $x_0 = \max\{x | (x, y) \in S_{min}\}$ to $x_1 = \infty$. For any given position of l, let $P_l = \{p_1, \ldots, p_i\}$ be the set of blue points to the left of l. We maintain a balanced binary search tree T over P_l, indexed by the y-coordinates of its elements. The intersection of ST_t with l is a single point q_i^t, which is the highest point on l that lies above at most t points of P_l. That is, q_i^t is the $(t+1)$-th smallest indexed entry in T, which we record. As we move l from left to right, q_i^t can only change when l intersects a point $p_i \in B_{NE}$. When we cross the point p_{i+1} (called *event point*), we insert it into T and, for each $t \leq k$, we have two cases.

1. If p_{i+1} is higher than q_i^t, then the height of ST_t does not change.
2. If p_{i+1} is lower than $q_i^t, q_i^{t-1}, \ldots, q_i^s$, for some $0 \leq s \leq t$, then q_i^t is set to q_i^{t-1} (which is done in $O(1)$ time), and ST_t moves to the height of this entry. This update is repeated by setting q_i^{t-1} to q_i^{t-2} and so on down to q_i^{s+1}. Finally, q_i^s is set to p_{i+1}.

If ST_t changes when sweeping over p_{i+1}, we also record a point r_i^t whose x-coordinate is that of p_{i+1} and whose y-coordinate is that of the updated q_i^t. Let B^t be the set of all such points q_i^t, r_i^t recorded during this process for all $t \leq k, i = 1, \ldots, m$. It is not hard to argue that $\cup_{t \leq k} ST_t \subseteq B^t$. Moreover, $|B^t| \leq m$ holds, as a point is added to Q only when the sweep line crosses a point of B_{NE}. Finally, note that we encountered $O(m)$ event points p_i. Inserting each of them into T requires $O(\log m)$ time, and $O(k)$ extra time is needed q_i^t, r_i^t pointer updates. Hence, the running time bound follows. □

See Figure 2 for an illustration of the proof of Lemma 1.

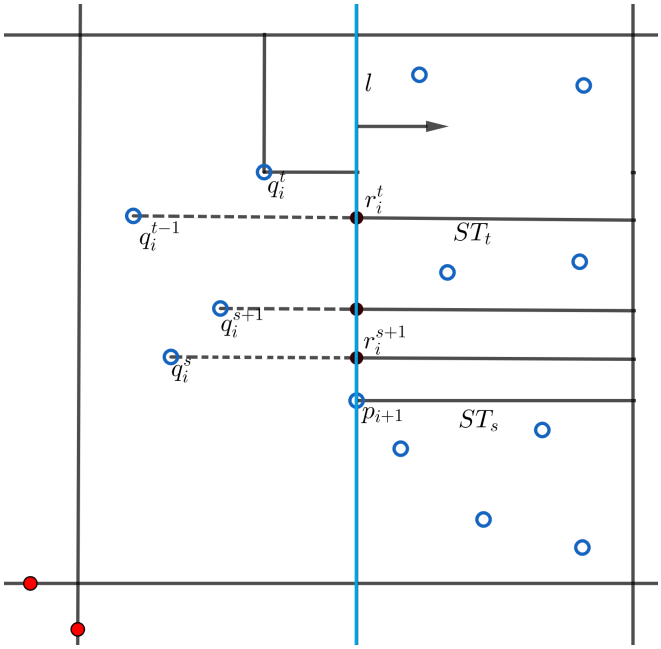

Figure 2. While sweeping vertical line l over point p_{i+1}, the staircase ST_t is updated.

Instead of computing the ST_t's for each partition of k, we compute all of them in $O(m \log m + mk)$ time before considering such partitions. Then we only need to solve a staircase problem in $O(m)$ time for each of the $O(k^7)$ partitions, giving us the following result.

Theorem 1. *Given two point sets, $R : |R| = n$ and $B : |B| = m$, as well as an integer $k \geq 0$, the largest rectangle enclosing R and containing up to k points in B can be computed in $O(k^7 m + m \log m + n)$ time.*

2.2. A Closer Look at the Number of Candidate Partitions of k

In the previous section, we reduced the running time by a factor of $\log m$. However, it seems hard to further improve this bound given the high number of partitions of k. Thus, in this subsection, we show how to reduce the number of candidate partitions of k, to further improve the running time bound.

To do that, we first compute all the t-level staircases $ST_t, 0 \leq t \leq k$ as described in the previous section. We then consider the blue points in 4 pairs of adjacent regions, e.g., N and NE. That is, we suppose the total number of outliers coming from $B_N \cup B_{NE}$, denoted $k_{NNE} = k_N + k_{NE}$, is fixed. Similarly, we suppose $k_{ESE} = k_E + k_{SE}, k_{SSW} = k_S + k_{SW}, k_{WNW} = k_W + k_{NW}$ are fixed. Let $ST(Q) = \cup_{t=1}^{k} ST_t(Q)$, for any quadrant Q. From now on, we focus on the N and NE regions and, for simplicity, we denote $ST(NE)$ as simply ST and $ST_t(NE)$ as simply ST_t.

We notice that even though any points of any t-th level staircase, $t \leq k_{NE}$, may be a corner for a candidate rectangle, most of these rectangles can be discarded as they are guaranteed to be smaller than the optimal rectangle.

Definition 3. *For every pair $P = (P_1, P_2)$ of regions and every integer $t : 0 \leq t \leq k$, denote by S_t^P the set of pairs $(p, q) \in (B_{P_1} \cup B_{P_2})^2$ such that p is the top support and q is the right support for an optimal solution, among all rectangles containing t blue points from $B_{P_1} \cup B_{P_2}$.*

From now on, for simplicity, we are going to remove the superscript and simply write S_t, e.g., $S_{k_{NNE}}$ instead of $S_{k_{NNE}}^{NNE}$. For every t, we store S_t as an array.

The goal is to compute $S_{k_{NNE}}$. Refer to Figure 3 for an illustration. Suppose we have already computed all $S_{t'} : t' < k_{NNE}$. Sweep $B_N \cup B_{NE}$ with a horizontal line l_H going upwards, starting at the $k_{NNE} + 1$-th lowest blue point in $B_N \cup B_{NE}$. For every blue point p encountered as top support, let $below(p)$ be the highest point in B_N below p, and $above(p)$

be the lowest point in B_N above p. For every $p \in B_{NE}$, let $rb(p)$ be the leftmost point in B_{NE} to the right of p and below p. Let t_N be the blue point count below p from B_N. Furthermore, let t be the number of points dominated by p from $B_N \cup B_{NE}$.

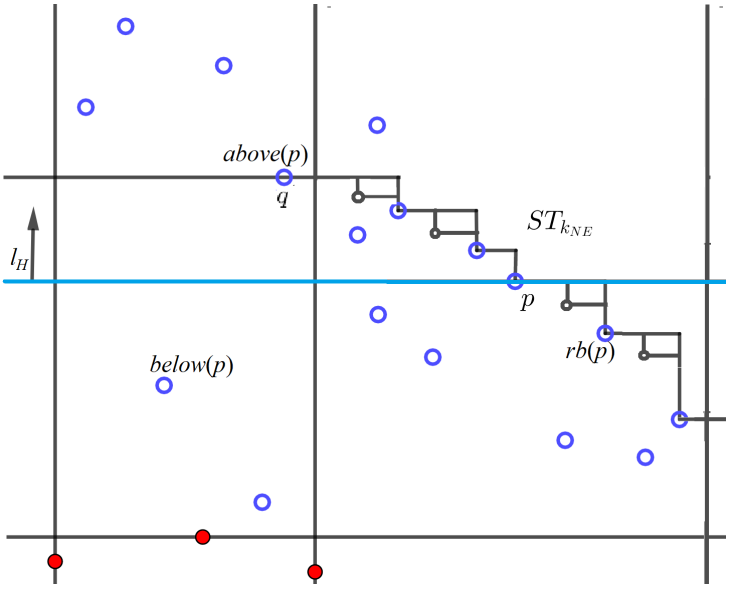

Figure 3. $B_N \cup B_{NNE}$ is swept with a horizontal line l_H sliding upwards from the lowest blue point. The set $S_{k_{NNE}}$ is updated at every blue point p encountered. Staircase points that are not blue points are marked with black dots.

First, assume $p \in B_{NE}$ and let $t_{NE} = t - t_N$ be the number of points dominated by p from B_{NE}, i.e., $p \in ST_{t_{NE}}$. When sweeping the next blue point q, we consider the following cases.

Case 1. If q is to the right of p, then q is below $above(p)$ but dominates p, the points dominated by p, and the points in $T(p,q) = \{s \in B_{NE} \cap ST_{t_{NE}} | x(p) < x(s) < x(q)\}$ (Figure 4). If $t + t_{pq} = k_{NNE}$, then we add $(q, rb(q))$ to $S_{t+t_{pq}}$, where $t_{pq} = 1 + |T(p,q)|$.

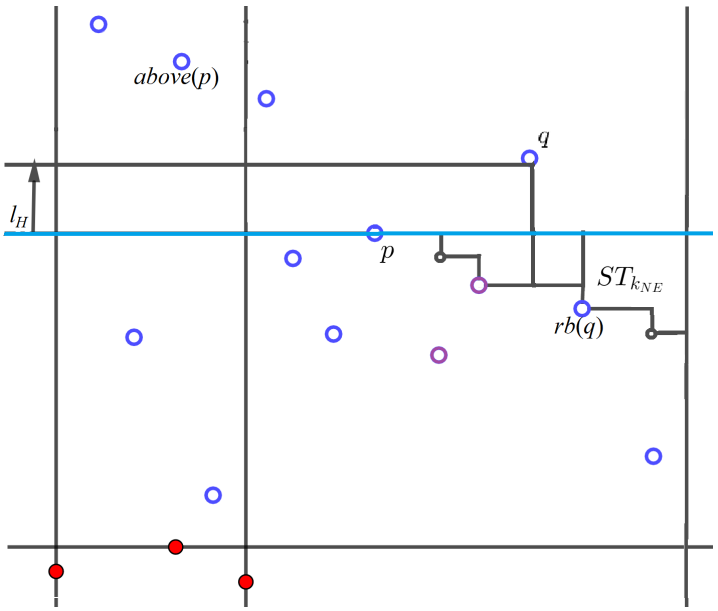

Figure 4. Case 1. $p \in B_{NE}$, q to the right of p. The purple empty dots denote $T(p,q)$.

Case 2. If $q \in B_{NE}$ and q is to the left of p, then q is below $above(p)$ (otherwise $(p,q) \notin S_{k_{NNE}}$), but dominates the points dominated by p, except the ones in $U(q,p) = \{s \in$

$B_{NE}|x(q) < x(s) < x(p), y(s) < y(p)\}$ (Figure 5). For each $s \in U(q,p)$, if $t - i(s) = k_{NNE}$, then we add (q,s) to $S_{t-i(s)}$, where $i(s)$ is the index of s in $U(q,p)$ in decreasing order of X coordinates. Finally, if $t = k_{NNE}$, then we add (q,p) to S_t.

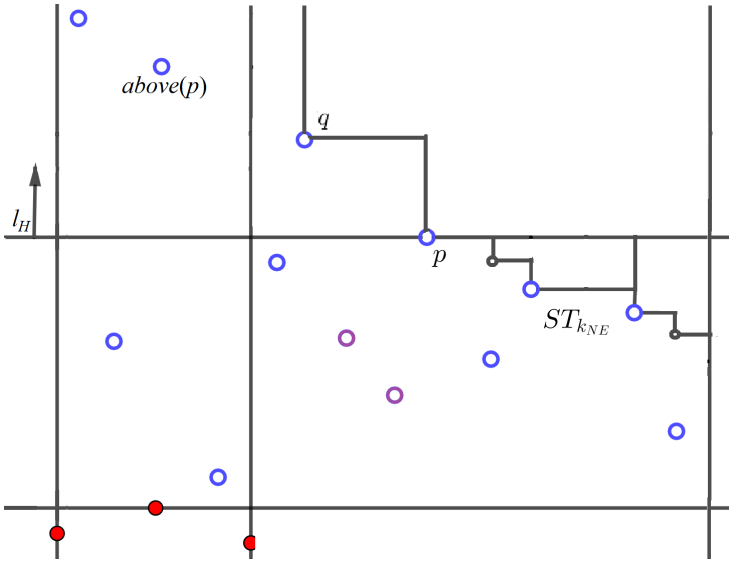

Figure 5. Case 2. $p \in B_{NE}$, $q \in B_{NE}$, and q is to the left of p. The purple empty dots denote $U(q,p)$.

Case 3. If $q \in B_N$ then, for each $s \in U(p) = \{s \in B_{NE}|x(s) < x(p), y(s) < y(p)\}$ such that $t - i(s) = k_{NNE}$, we add (q,s) to $S_{t-i(s)}$, where $i(s)$ is the index of s in $U(p)$ in decreasing order of X coordinates. Finally, if $t = k_{NNE}$, then we add (q,p) to S_t (Figure 6).

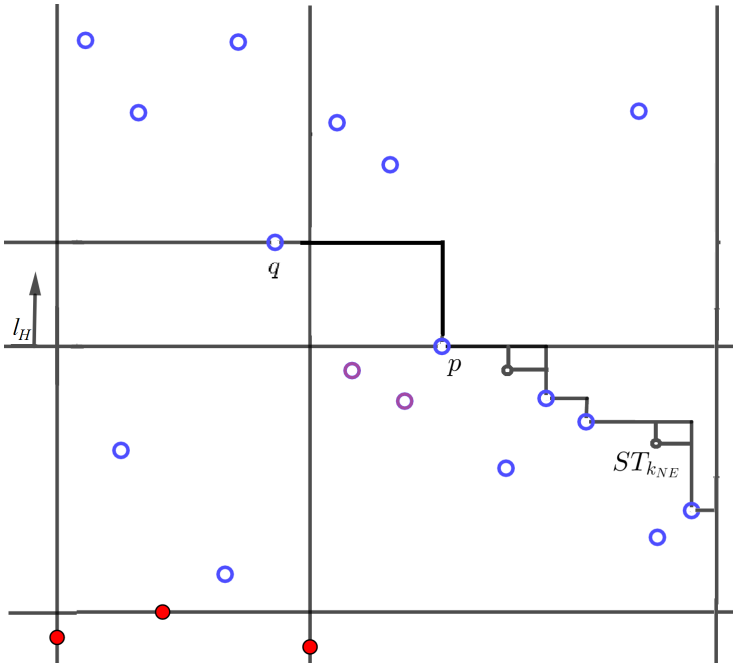

Figure 6. Case 3. $p \in B_{NE}, q \in B_N$. The purple empty dots denote $U(p)$.

Now assume $p \in B_N$ and let t_{NE} be the largest t' such that all points in any $ST_{t'}$ are below p. Let $b(p)$ be the leftmost point of $ST_{t_{NE}}$ below p. When sweeping the next blue point q, we consider the following cases.

Case 4. If q is to the right of $b(p)$ then, if $t = k_{NNE} - 1$, we add $(q, rb(q))$ to S_{t+1}. For each $s \in U(q)$ such that $t - i(s) = k_{NNE}$, we also add (q,s) to $S_{t-i(s)}$ (Figure 7).

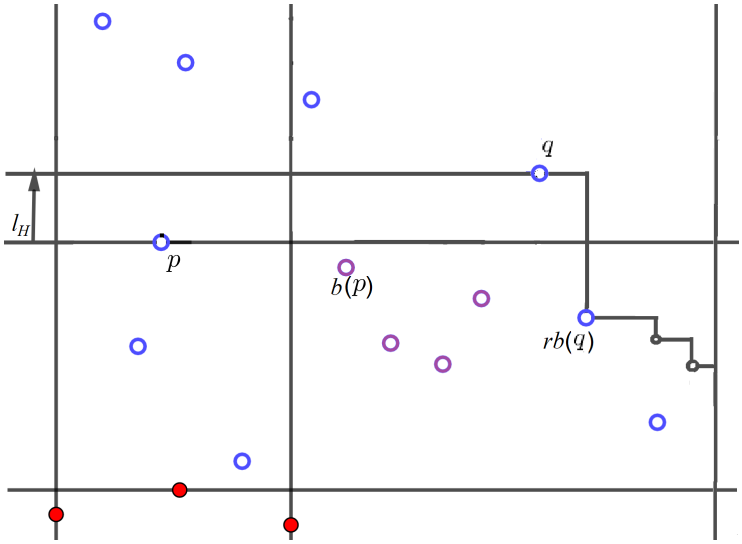

Figure 7. Case 4. $p \in B_N$, q is to the right of $b(p)$. The purple empty dots denote $U(q)$.

Case 5. If $q \in B_{NE}$ and q is to the left of $b(p)$, then, if $t = k_{NNE} - 1$, we add (q, s) to S_{t+1} for every $(p, s) \in S_t$ (Figure 8).

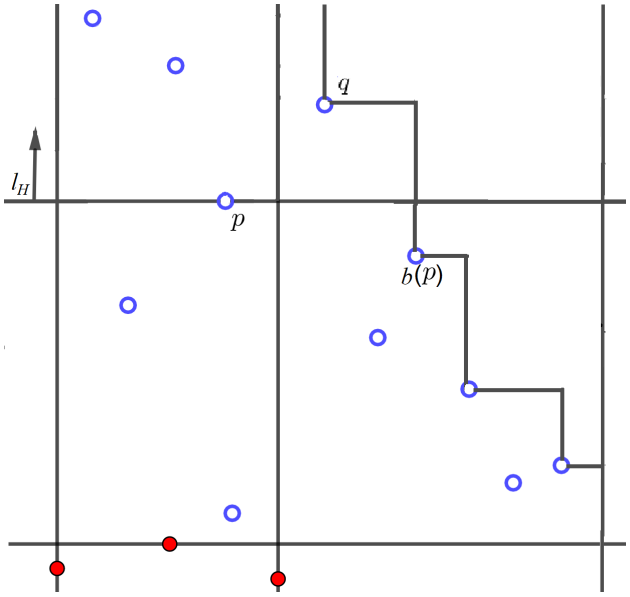

Figure 8. Case 5. $p \in B_N$, $q \in B_{NE}$ and q is to the left of $b(p)$.

Case 6. If $q \in B_N$, then, if $t = k_{NNE} - 1$, we add (q, s) to S_{t+1} for every $(p, s) \in S_t$ (Figure 9).

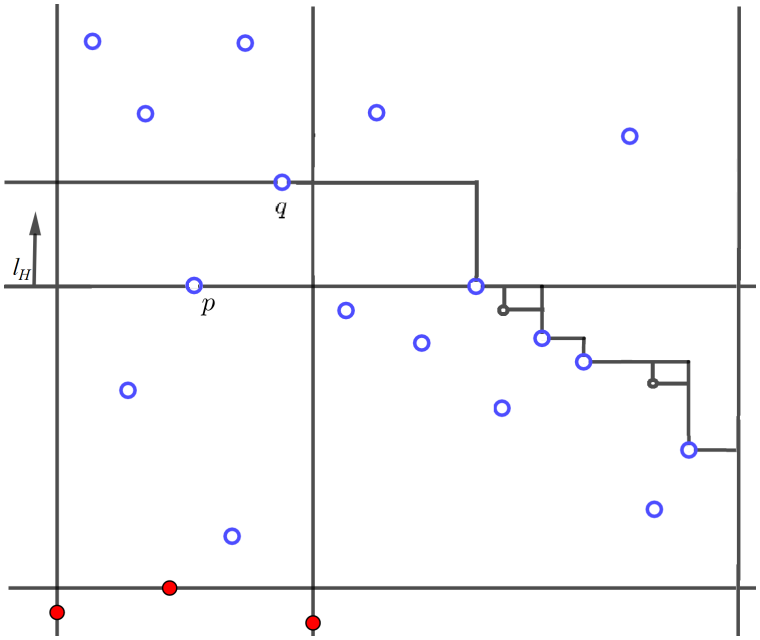

Figure 9. Case 6. $p \in B_N, q \in B_N$.

The following lemma puts an upper bound on the storage required by $S_{k_{NNE}}$.

Lemma 2. $|S_{k_{NNE}}| = O(m)$.

Proof. In case 1, we only add one pair to $S_{k_{NNE}}$. In case 2, even though we consider $|U(q,p)|$ points, we only add the pair (q,s) such that $t - i(s) = k_{NNE}$. Similarly, in cases 3 and 4 we only add the pair $(q,s) : t - i(s) = k_{NNE}$, even though we consider $|U(p)|$ (resp., $|U(q)|$) points. In case 5, we add at most $|ST_{t_{NE}}|$ pairs if $t = k_{NNE} - 1$. However, note that for the subsequent point q' swept, we would have a larger number t' of blue points in $B_N \cup B_{NE}$ dominated by q'. Thus, we only add at most $|ST_{t_{NE}}| = O(m)$ pairs once. Similarly, in case 6 we only add $O(m)$ pairs once. □

The following lemma states the running time of the aforementioned sweeping algorithm.

Lemma 3. *For any $t : 0 \leq t \leq k$, the horizontal line sweeping described above takes $O(m \log m)$ time.*

Proof. We store the blue points in $B_N \cup B_{NE}$ in two balanced binary search trees X, Y, indexed by X (resp., Y) coordinates. Thus, for each blue point p swept, we require $O(\log m)$ time. We require an extra $O(\log m)$ time to compute $above(p)$, $below(p)$, and $rb(p)$. In case 1, note that we can compute t_{pq} by finding the position of q in the X-sorted order of $ST_{k_{NE}}$, and thus the number of blue points $s : x(p) < x(s) < x(q)$, in $O(\log m)$ time, since $ST_{k_{NE}}$ is maintained as a binary search tree. Thus, we only require an extra $O(\log m)$ time to handle case 1. In cases 2 and 4, note that we only need to add (q,s) to $S_{k_{NNE}}$ if $i(s) = t - k_{NNE}$, so we query for s in X using $O(\log m)$ time. Similarly, in case 3 we only add (q,p) to $S_{k_{NNE}}$ if $i(s) = t - k_{NNE}$, so we query X for s in $O(\log m)$ time. Now in cases 5 and 6 we spend $O(m)$ time to traverse S_t, since we store S_t as an array for any t, but they only occur once, so this gives us $O(m)$ total time. In every case, since S_t is an array, adding a pair to S_t takes $O(1)$ time. Since we sweep $O(m)$ blue points, the result follows. □

Corollary 1. *For any pair of quadrants, we compute S_t in $O(km \log m)$ time for all $t : 0 \leq t \leq k$.*

We reduce the number of candidate partitions of k from $O(k^7)$ to $O(k^3)$ as follows. By writing $k = k_{NNE} + k_{ESE} + k_{SSW} + k_{WNW}$, we can deduce $k_{NNE} = k - k_{WNW} - k_{ESE} - k_{SSW}$ for every combination of $k_{WNW}, k_{ESE}, k_{SSW}$. Therefore, there are $O(k^3)$ such combinations.

Initially, we compute S_t^Q for every quadrant Q and $0 \leq t \leq k$. Then, for each combination $(k_1, k_2, k_3), k_1, k_2, k_3 = 0, \ldots k, k_1 + k_2 + k_3 \leq k$, we set $k_4 = k - k_1 - k_2 - k_3$ and solve the staircase problem in [2] with the pairs $S_{t_1}^{WNW} \cup S_{t_2}^{ESE} \cup S_{t_3}^{SSW} \cup S_{k_4}^{NNE}$ as pairs of supports. Each staircase problem takes $O(m)$ time to solve, so we require $O(k^3 m)$ time for all candidate partitions of k. Putting this together with the result in Lemma 1, we get the following result.

Theorem 2. *Given two point sets, $R : |R| = n$ and $B : |B| = m$, as well as an integer $k \geq 0$, the MBSR-O for R and B with k outliers can be computed in $O(k^3 m + km \log m + n)$ time.*

3. Finding the Largest Axis Aligned Rectangle Enclosing R and Avoiding Unit Circles

In this extension, B consists of S_{min}-disjoint unit circles, and the goal is to find the largest axis-aligned rectangle that avoids all circles while enclosing R. We call such rectangle an MBSR among circles or MBSR-C.

Again, we discard blue circles intersecting S_{min} from consideration, as they cannot be avoided.

One may wonder whether the reduction in [3] for finding the largest separating rectangle among axis-aligned rectangles can be tailored to MBSR-C. However, it can be shown that it does not always work. If we let $C_{NW}, C_{SW}, C_{SE}, C_{NE}$ be circles in the regions $B_{NW}, B_{SW}, B_{SE}, B_{NE}$, pick any point p on the quadrant of C_{NW} that is the closest to S_{min}, and add it to B', then any rectangle enclosing R, avoiding B', and top or right-bounded by p will intersect C_{NW}. See Figure 10 for a depiction of why this is the case.

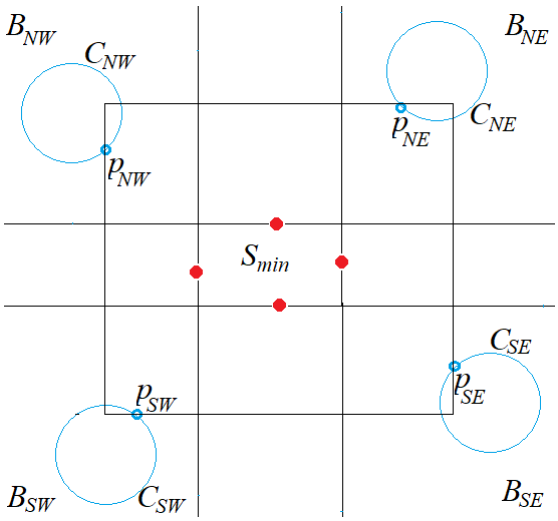

Figure 10. If the MBSR among R and B' were bounded by a point $p \in C$, it would intersect C.

We call a *candidate separating rectangle* (CSR) a rectangle that encloses R and cannot be extended in any direction without intersecting some circle. Note that a CSR may touch a circle either at a corner or at an edge. If it is bounded at an edge, then that edge is fixed in terms of X or Y coordinate and the arc it touches at each endpoint of the edge is uniquely determined (Figure 11). On the other hand, if it is bounded at a corner, then the corner can be slid along the appropriate arc of the circle (Figure 12). Each position of the corner determines the X or Y coordinates of its two adjacent edges, and thus the arcs pinning the two adjacent corners, if any.

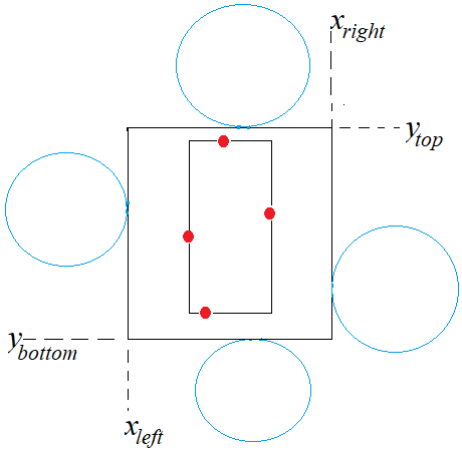

Figure 11. A circle bounding a rectangle at an edge makes that edge fixed in terms of either X or Y coordinate.

We say that an edge e of a CSR is *pinned* by a circle C if C touches the interior of e.

A horizontal (resp., vertical) edge e is said to be *fixed* by two circles C_1, C_2 in terms of Y (resp., X) coordinate, if:

(1) the ends of e are on C_1 and C_2, respectively, and
(2) changing its Y (resp., X) coordinate would result in either e intersecting C_1 or C_2 or failing to touch both C_1 and C_2.

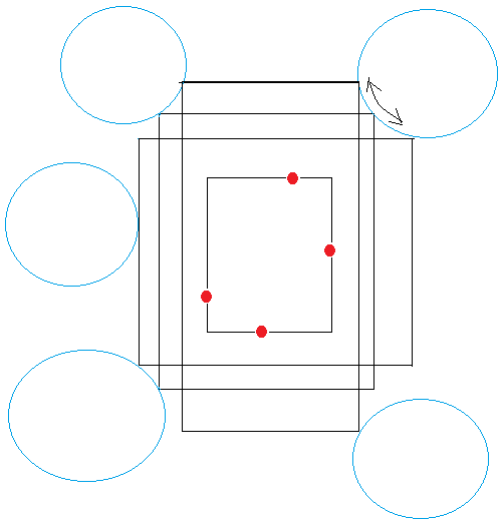

Figure 12. A circle bounding the rectangle at a corner allows the corner to slide along the arc. Each position of the corner uniquely determines its two adjacent edges.

3.1. A Description of All Cases in Which a CSR Can Be Found

We consider all the cases in which a CSR can be found, based on the number of edges pinned by circles.

Case 1. Three edges pinned by circles (Figure 13). In this case, we extend the the fourth edge outward from S_{min} until it touches a circle. Hence, the CSR is uniquely determined.

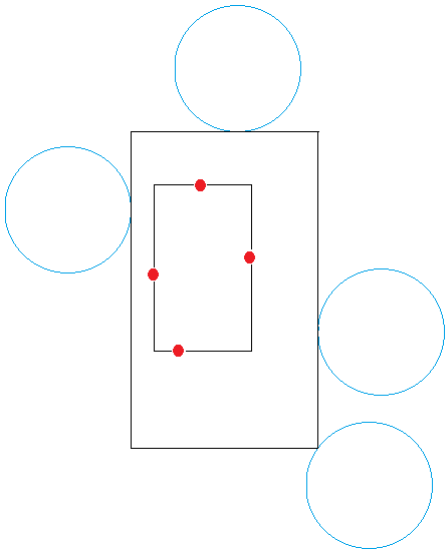

Figure 13. Case 1: Three circles pin the CSR on edges, which uniquely determines its fourth edge.

Case 2. Two edges are pinned by two circles C_1, C_2. In this case, we further distinguish the following subcases.

Case 2.1. Two adjacent edges are pinned by C_1, C_2. Note that their common corner q is fixed. We extend one of the edges by moving its other end p away from q until it touches a circle C_3, and then extend the third edge until it touches a circle C_4 at a point r (Figure 14). The resulting CSR is unique.

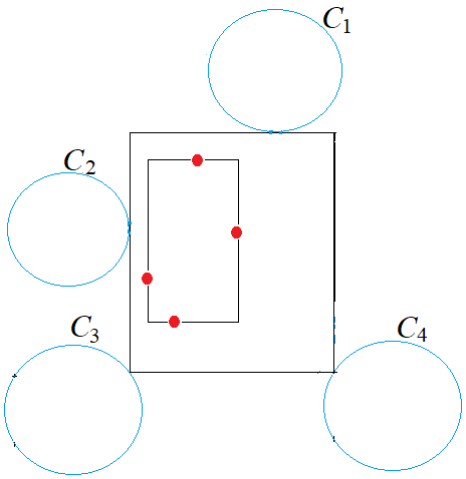

Figure 14. Case 2.1: Two circles C_1, C_2 pin two adjacent edges of the CSR. When extending one of the edges until it touches a circle C_3, the resulting CSR is unique.

Case 2.2. Two adjacent edges are pinned by C_1, C_2. We extend one of the edges by moving its other end p away from q, until the orthogonal line through p touches a circle C_3 at a point r (see Figure 15). While moving p, the point r can slide along one or more circles in the same quadrant, giving an infinite number of CSRs.

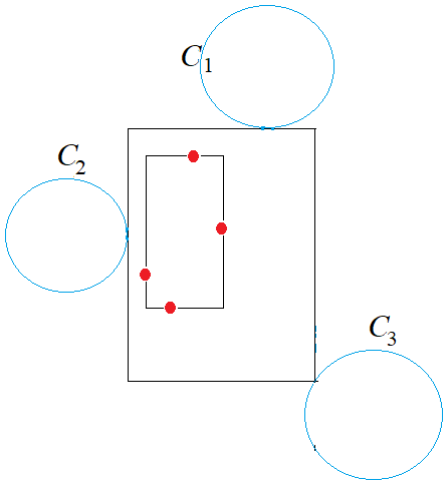

Figure 15. Case 2.2: Two circles C_1, C_2 pin the CSR on the north and west edges. The SE corner may slide along a circle C_3.

Case 2.3. Two opposite edges are pinned by C_1, C_2. We slide the other two edges outward from S_{min} until each of them touches some circle (see Figure 16). This gives us a unique CSR.

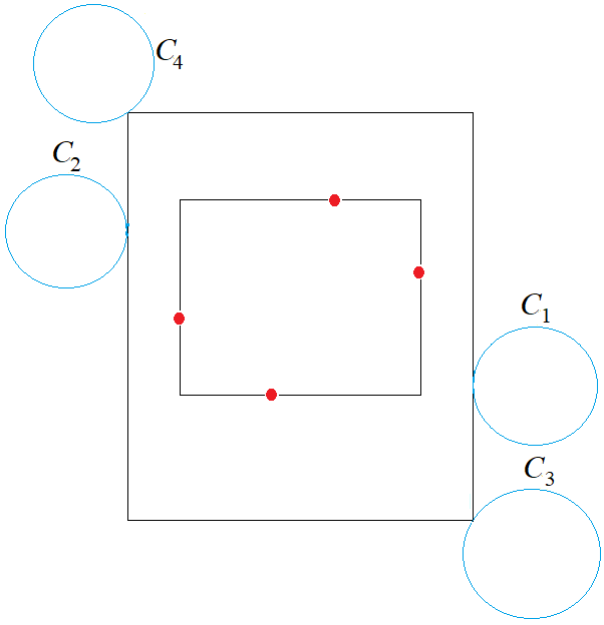

Figure 16. Case 2.3: Two circles C_1, C_2 pin the CSR on the east and west edges. The other two edges are uniquely determined.

Case 3. One edge e is pinned by a circle C_1. We have the following subcases.

Case 3.1. When e is extended in both directions, it touches two circles C_2, C_3 (Figure 17). We then slide the fourth edge outward from S_{min} until it touches a circle C_4 and we have a unique CSR.

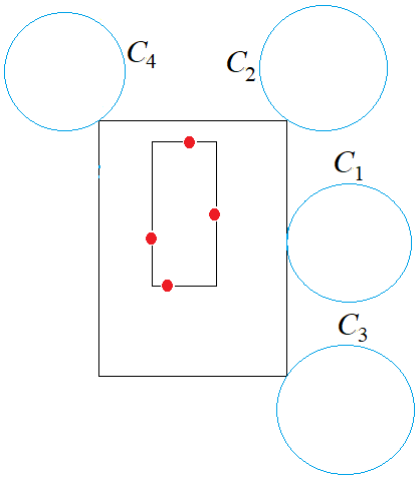

Figure 17. Case 3.1: Circle C_1 supports the CSR on the east edge, which, when extended in both directions, it eventually touches circles C_2, C_3.

Case 3.2. When e is extended in both directions, the orthogonal line through one of the ends p touches a circle C_2 at point q (Figure 18). While moving p, q can slide along one or more circles in the same quadrant, yielding an infinite number of CSRs. After establishing the position of q, we slide the fourth edge away from S_{min} until it touches a circle C_3.

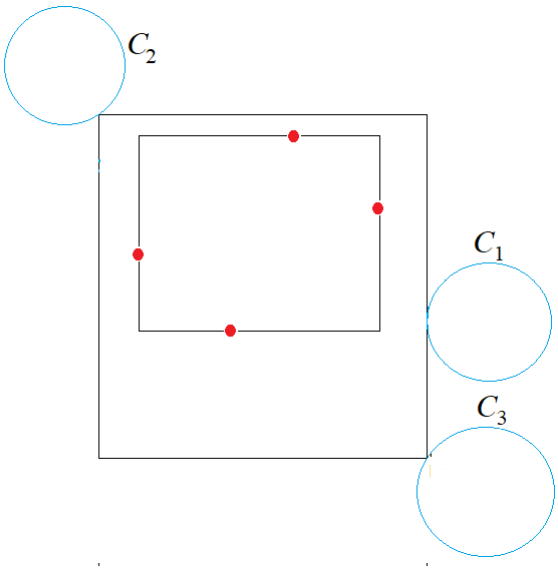

Figure 18. Case 3.2: The CSR is supported by C_1 on the east edge. One of the adjacent edges is uniquely determined, while the opposite corner may slide along a circle C_2.

Case 4. No edge is pinned by any circle. In this case, all corners can slide along circles until one of the edges becomes pinned by some circle, giving an infinite number of CSRs. Suppose the position of a corner p along a circle $C_1 \in B_{NE}$ is known. We consider the following subcases.

Case 4.1, while extending the CSR in the two directions away from p, the CSR touches a circle in B_{SE} or B_{NW} at some point q before touching any circle in B_{SW} (Figure 19). The other two corners are determined by sliding the edge opposite to pq outwards until it touches a circle at some point r. In this case, the CSR is uniquely defined.

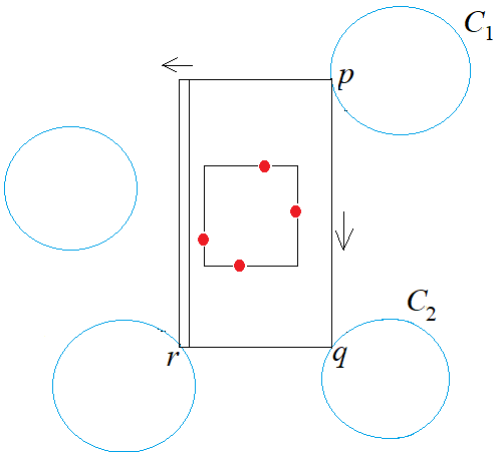

Figure 19. Case 4.1: No edge is pinned by any circle, and the position of p on $C_1 \in NE$ is known. While sliding the CSR left-downward from S_{min}, we first touch circle C_2 at corner q adjacent to p. The other two corners are uniquely determined.

Case 4.2, while extending the CSR in the two directions away from p, the first circle that CSR touches, at a point q, is located in B_{SW} (Figure 20). This gives us an infinite number of CSRs.

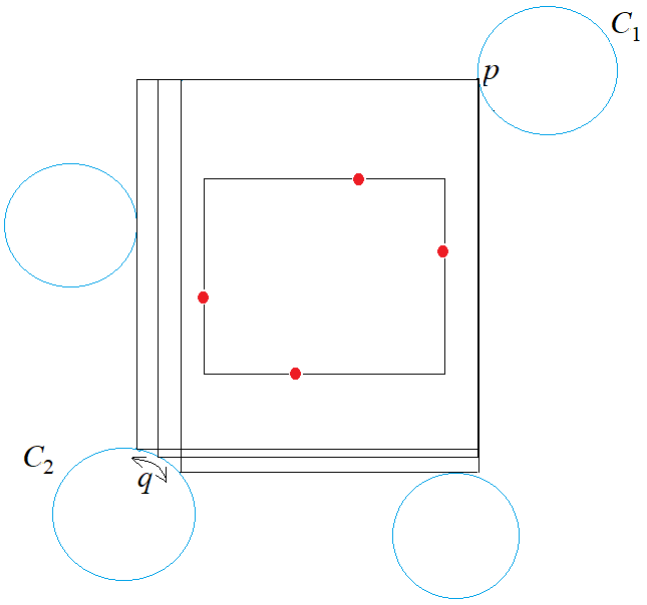

Figure 20. Case 4.2: No edge is pinned by any circle, and the position of p on $C_1 \in NE$ quadrant is known. While sliding the CSR left-downward from S_{min}, we first touch circle C_2 at a corner q opposite to p. There are an infinite number of CSRs in this case.

3.2. Dominating Envelopes

Definition 4. *The dominating envelope of a corner region B_i is a curve $\mathcal{C}$ satisfying the following:*

1. *B_i fully contains $\mathcal{C}$,*
2. *$\forall p \in \mathcal{C}$, the rectangle cornered at p and the closest corner of S_{min} from p is empty, and*
3. *property (2) no longer holds if one extends $\mathcal{C}$ away from S_{min}.*

Note that $\mathcal{C}$ is a sequence of horizontal and vertical segments, circle arcs, as well as a horizontal and a vertical infinite ray (see Figure 21). We shall reveal the use of dominating envelopes later.

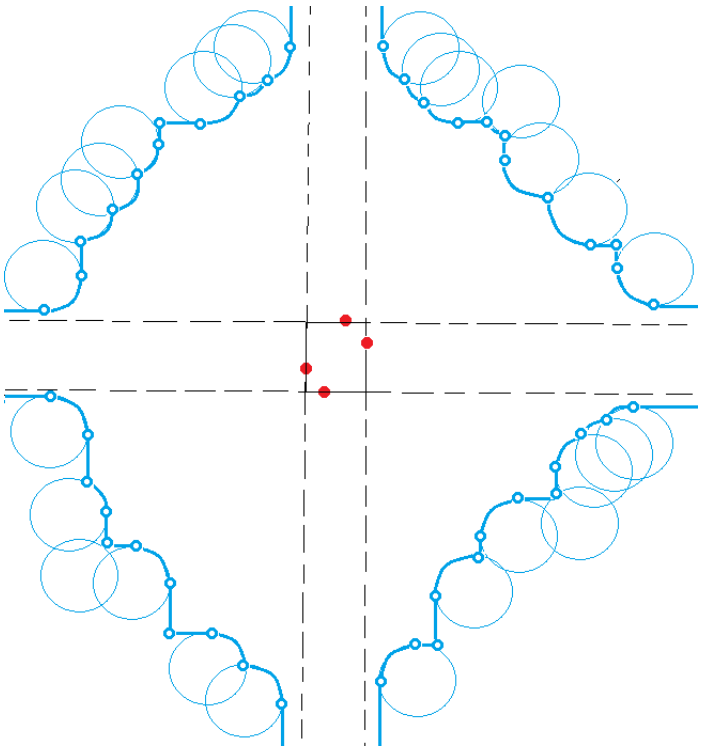

Figure 21. In each region, the circles define a dominating envelope $\mathcal{C}$, which is a sequence of arcs and horizontal or vertical segments. Two consecutive arcs or segments define a breakpoint on $\mathcal{C}$.

The dominating envelope changes direction at *breakpoints*, which can be between two consecutive arcs, segments, or an infinite ray. Every two consecutive breakpoints define a *range of motion* for a CSR corner.

A breakpoint p is said to be a *corner breakpoint*, if a CSR cornered at p cannot be extended away from S_{min} in all directions without crossing some circle, even if its other corners are not located on any envelope.

To compute the dominating envelope $\mathcal{C}$ of B_{NE}, we do the following. First, sort the circles by X coordinate of their centers. Let p be the current breakpoint (initially, the first breakpoint is the left endpoint l of the left-most circle, with a vertical infinite ray upwards from l). For each two adjacent circles $C_1(c_1, 1), C_2(c_2, 1)$, depending on the relative positions of c_1, c_2, we do the following.

Case A. $c_2 \in C_1$. In this case, we add the lower intersection between C_1 and C_2 as a new corner breakpoint q, along with the arc pq of C_1.

Case B. $y(c_1) - 1 \leq y(c_2) \leq y(c_1)$ and $x(c_2) > x(c_1) + 1$. We add a breakpoint q at the bottom of C_1, the arc pq of C_1, a corner breakpoint r at the intersection between the horizontal through q and C_2, and the line segment qr.

Case C. $x(c_1) + 1 \leq x(c_2) \leq x(c_1) + 1$ and $y(c_2) < y(c_1) - 1$. We add a breakpoint r at the left endpoint of C_2, a corner breakpoint q at the intersection between the vertical through r and C_1, the line segment qr, and the arc pq of C_1.

Case D. $x(c_2) > x(c_1) + 1$ and $y(c_2) < y(c_1) - 1$. We add the breakpoints q at the bottom of C_1, r at the left end of C_2, the *corner breakpoint* s at the intersection between the horizontal through q and the vertical through r, the arc pq of C_1, and the segments qs and sr.

We then set p to the rightmost breakpoint added and repeat the process for the next pair of adjacent circles. Finally, for the last circle, we add an arc from p to its bottom b, the breakpoint b, and then a horizontal infinite ray emanating from b to the right. In each case, we say that a pair of circles (C_1, C_2) *defines* breakpoints $p_1, \ldots, p_k$, if all of $p_1, \ldots, p_k$ are added as breakpoints in the process described above. Similarly, we say (C_1, C_2) *defines* arcs $a_1, \ldots, a_k$, if all of $a_1, \ldots, a_k$ are added as arcs in the process. See Figure 22 for an illustration of this process.

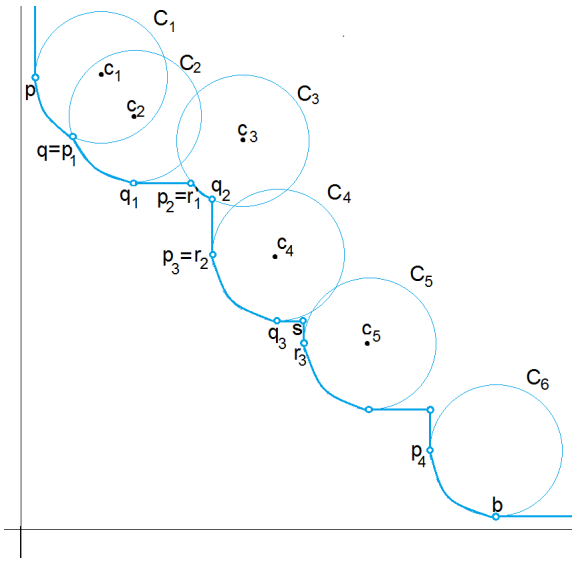

Figure 22. Each pair of adjacent circles gives a different case.

Note that deciding the case a circle belongs to can be done in constant time.

3.3. Finding an Optimal Solution in Each Case

We first slide the edges of S_{min} outward until each of them touches a circle. Since we assumed no unbounded solutions, we are guaranteed that every edge will eventually hit a circle. Denote by S_{max} the resulting rectangle and discard the region outside S_{max} from the dominating envelopes of all quadrants. The endpoints of the resulting envelopes are also counted as breakpoints. We also sort the blue circles by the X coordinate and then (to break ties) by the Y coordinate of their centers.

Now we give a specific algorithm to compute the MBSR-C in each of the cases listed in Section 3.1.

Case 1. We consider all corner breakpoints that are defined by pairs of adjacent circles in Case D, and add them to a set B'. We also consider all arcs pq of circles that are part of pairs in Case D (p to the west of q), the vertical line l_p through p and the horizontal line l_q through q, and add $l_p \cap l_q$ to B'. We then find the largest rectangle S^* enclosing R and containing the fewest points in B' using the algorithm in [1] in $O(m+n)$ time. It is easy to check that S^* is an optimal solution for Case 1, since any circle containing points in B' intersects a circle in B. Thus, Case 1 can be done in $O(m+n)$ time.

Case 2. Assume wlog that two circles pin the north and the west edges of a CSR. The cases where the two circles define a different pair of adjacent edges of a CSR can be handled in a similar fashion.

If we are in Case 2.1, we consider all corner breakpoints q defined by pairs of adjacent circles in Case D, as well as the intersection between the south horizontal tangent t_H to the eastmost circle in B_{NE} and the east vertical tangent t_V to the northmost circle in B_{NW} south of t_H. For every such point, the north and west edges are fixed, and we either find the south edge by extending the west edge southwards until it hits a blue circle, or the east edge by extending the north edge eastwards until it hits a blue circle. In both approaches, the fourth edge is uniquely determined. For each circle in $C \in B_{NE}$, we store pointers to the northmost circle in B_{NW} south of C and to the eastmost circle in B_{SE} west of C, as well as similar pointers for the other quadrants and directions, Thus, once the first two edges are fixed, we can find the third and fourth edges in $O(1)$ time. Since there are $O(m)$ circles in Case 2.1 and they all can be found in $O(m)$ time, Case 2.1 can be solved in $O(m)$ time.

For Case 2.2, we consider all points q as in Case 2.1. Having selected such point q defined by two circles $C_1 \in B_{NE} \cup B_{NW}$ and $C_2 \in B_{NW} \cup B_{SW}$, we consider the dominating envelope of B_{SE} starting from the east tangent to C_1 or the east edge of S_{min}, whichever is eastmost, and ending at the south tangent to C_2 or the south edge of S_{min}, whichever is

southmost. This gives us a range of motion for the SE corner r of the CSR spanning $O(m)$ circle arcs. For each such arc, we find the optimal CSR in $O(1)$ time as we shall prove in the next section. Since there are $O(m)$ choices of q, we handle Case 2.2 in $O(m^2)$ time.

As for Case 2.3, note that the pairs of circles defining the NW and the SE corners, respectively, must belong to a dominating envelope. We scan the dominating envelope of B_{NW} for pairs of circles C_1, C_2 in Cases B and C and, for each such pair, we scan the dominating envelope of B_{SE} for pairs of circles in Cases B and C, starting from the east tangent to C_1 or the east edge of S_{min}, whichever is eastmost, and ending at the south tangent to C_2 or the south edge of S_{min}, whichever is southmost. Once these pairs are established, the CSR is determined. Since scanning each dominating envelope takes $O(m)$ time, we handle Case 2.3 in $O(m^2)$ time.

Case 3. Assume wlog that a circle C pins the east edge of the CSR. The cases where the circle define a different edge of the CSR can be handled in a similar fashion.

For Case 3.1, we scan the dominating envelope of B_{NE} (similarly, B_{SE}) for pairs of circles (C_1, C) in cases C and D, (C is the rightmost circle of the pair). For every such pair of circles in B_{NE}, we consider all circles $C_2 \in B_{SE}$ that are intersected by the west vertical tangent to C. It is possible that some of these circles were already considered for a previous pair of circles in B_{NE}, so we may have to consider $O(m^2)$ triplets of circles (C, C_1, C_2). We also traverse the circles in B_E in increasing X order of their centers. Denote by C the current circle. We consider the sequences of circles $C_{NE} \in B_{NE}$ and $C_2 \in B_{SE}$ that are intersected by the west tangent to C. Since these sequences may include circles already considered for a previous circle in B_E, we may need to spend $O(m^2)$ to find all such triplets (C, C_1, C_2) for which there exists a vertical line intersecting both C_1, C_2. Once a triplet is established, the west, north, and south edges are established, and the west edge can be determined in $O(1)$ time by extending the north or the south edge until it hits a circle. Thus, Case 3.1 requires $O(m^2)$ time.

For Case 3.2, we scan the dominating envelope of B_{SE} (similarly, B_{NE}) for pairs of circles (C_1, C) in cases C and D (C is the rightmost circle of the pair). We also traverse the circles $C \in B_E$ in increasing X order and consider the sequences of circles $C_1 \in B_{SE}$ that are intersected by the west tangent to C. For each pair (C_1, C), the east and south edges of the CSR are defined. This provides a range of eligible circles from the dominating envelope of B_{NW} such that the SW and NE corners of the CSR are not supported by any circle, and the NW corner slides along some circle arc. There are $O(m)$ (C_1, C) pairs and each of them gives $O(m)$ circles from B_{NW}. Hence, Case 3.2 takes $O(m^2)$ time.

Case 4. Consider all arcs defined by pairs of adjacent circles in one of the cases A, B, C, or D. Consider all arcs defined by pairs of adjacent circles. Each such arc a establishes the range of motion for the appropriate corner p of a CSR, say $[p_{start}, p_{end}]$ in X order. Suppose a belongs to a circle in B_{NE}, which establishes the range of motion of the NE corner p of the CSR. This gives us a range of sliding motion for the north and the east edges of the CSR, which are supported by two rays r_W, r_S shooting from p to the west and south, respectively. Since only the SW corner q may also slide along a circle, the west edge can be neither to the west of the first intersection $W(p)$ between r_W and a circle, nor to the west of the easternmost point in B_W of a circle. Similarly, the south edge can be neither be to the south of the first intersection $S(p)$ between r_S and a circle, nor to the south of the northernmost point in B_S of a circle. This gives a range of motion for q, which may span multiple circle arcs with X coordinates within the range $arcs(p_{start}, p_{end})$ = $[\min(X(W(S(p_{end})))$, $X(S(W(p_{end}))))$, $\max(X(W(S(p_{start})))$, $X(S(W(p_{start}))))]$. In fact, there are $O(m)$ arcs in the worst case, yielding $O(m^2)$ pairs of arcs for all possible pairs (p, q). By computing the pointers $west(p), south(p), east(p)$, and $north(p)$ for every p, from the dominating envelope, we can find each pair of arcs in $O(1)$ time, as we take them in X order. That is, we consider all arcs $[p_{start}, p_{end}]$ defined by pairs of adjacent circles in X order and, for each such arc, we compute the points $W(S(p_{start}))$, $S(W(p_{start}))$, $W(S(p_{end}))$, and $S(W(p_{end}))$, and then consider the arcs in B_{SW} with X coordinates within $arcs(p_{start}, p_{end})$.

Handling each pair of arcs in $O(1)$ time will be detailed in the next subsection. Thus, computing the optimal solution takes $O(m^2)$ time in Case 4.

3.4. Finding the CSR Once the Arcs Pinning Its Corners Are Selected

For each quadrant Q, let $\theta_Q \in [\alpha_Q, \beta_Q]$ be the angular position of the corner within the arc belonging to Q, say $C(c,1)$. Let $f(\theta_{NE}, \theta_{NW}, \theta_{SW}, \theta_{SE})$ denote the area of the CSR with the corners defined in terms of θ_Q as above. Our goal is to find the maximum of f over the feasible set of arguments, along with its arguments. First, assume $\theta_{SE}, \theta_{NW}, \theta_{SW}$ are fixed, with the left and bottom supports denoted as l, b (Figure 23). We refer to $f(\theta_{NE}, \theta_{NW}, \theta_{SW}, \theta_{SE})$ as simply $f_{NE}(\theta)$ (that is, refer to θ_{NE} as simply θ).

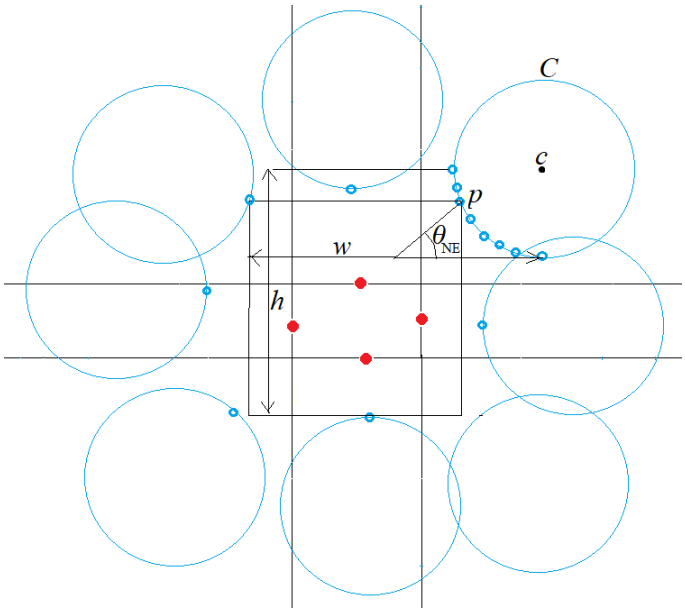

Figure 23. Area of the CSR cornered on circle $C(c,1)$ as a function $f_{NE}(\theta_{NE})$ of that corner's location.

Lemma 4. *f_{NE} has at most 3 maxima.*

Proof. We have

$$f_{NE}(\theta) = (w - \sin\theta) \cdot (h - \cos\theta), \tag{1}$$

where $w = x(c) - x(l)$ and $h = y(c) - y(b)$. For simplicity, assume that all circles are fully contained in some quadrant, so $w, h > 1$. Furthermore,

$$f'_{NE}(\theta) = w\sin\theta - h\cos\theta + \sin^2\theta - cos^2\theta. \tag{2}$$

Letting $x = \tan\theta$, we get

$$f'_{NE}(x) = \frac{wx - h}{\sqrt{1+x^2}} + \frac{x^2 - 1}{1 + x^2}, \tag{3}$$

so $f'_{NE}(x) = 0 \iff$
$(wx - h)\sqrt{1+x^2} = 1 - x^2 \iff$
$(wx - h)^2(1 + x^2) = (1 - x^2)^2 \iff$
$(w^2 - 1)x^4 - 2whx^3 + (w^2 + h^2 + 2)x^2 - 2whx + h^2 - 1 = 0 \iff$
$(w^2 - 1)x^2(x^2 - 1) - 2whx(x^2 - 1) + (h^2 - 1)(x^2 - 1) = 0 \iff$
$((w^2 - 1)x^2 - 2whx + h^2 - 1)(x^2 - 1) = 0,$
which solves to

$$x_1 = 1, \tag{4}$$

$$x_{2,3} = \frac{wh + -\sqrt{2(w^2 + h^2) - 1}}{w^2 - 1} \tag{5}$$

(we ignore negative roots since $x \geq 0$). Since $x = \tan\theta$, it follows that $\theta_1 = \frac{\pi}{4}, \theta_{2,3} = \arctan x_{2,3}$ are extrema for f_{NE}. That is, f_{NE} has no more than 3 maxima. □

Similarly,it follows that f_{NW}, f_{SW}, f_{SE} each have no more than 3 maxima. Note that the position of two opposite corners of a CSR, say NE and SW, determine the position of the other two corners. Since there are only two variables, f has at most 9 maxima we need to consider. We compute all these maxima in $O(1)$ time and then choose the one that gives the largest CSR.

Thus, we have proved the following result.

Theorem 3. *Given a set of n red points R and a set of m blue unit circles B with $|B| = m$, the MBSR-C among R and B can be computed in $O(m^2 + n)$ time.*

4. Conclusions and Future Work

We improve upon the existing result for the maximum bichromatic separating rectangle with outliers problem (MBSR-O) of $O(k^7 m \log m + n)$ and provide an $O(k^3 m + km \log m + n)$ time algorithm. We also consider the problem of finding the maximum bichromatic separating rectangle among unit circles (MBSR-C), for which we give an algorithm that takes $O(m^2 + n)$ time [14].

We leave for future consideration proving lower bounds and finding efficient approximation algorithms for MBSR-C and MBSR-O. Other interesting related problems would be finding the smallest and largest circle enclosing red points while avoiding blue circles. Finally, we leave for future work finding the largest rectangle separating red points from blue polygons, as well as solving the weighted version, in which red points have a positive weight, blue points have a negative weight, and the goal is to find the largest rectangle of maximum weight. This weighted version would have promising applications in circuit design, where the board defects may have various severity levels.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The author would like to thank Benjamin Raichel, Chenglin Fan, and Ovidiu Daescu for the useful discussions.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Armaselu, B.; Daescu, O. Maximum Area Rectangle Separating Red and Blue Points. *arXiv* **2017**, arXiv:1706.03268.
2. Armaselu, B.; Daescu, O. Maximum Area Rectangle Separating Red and Blue Points. In Proceedings of the Canadian Conference on Computational Geometry, Vancouver, BC, Canada, 3–5 August 2016; pp. 244–251.
3. Armaselu, B.; Daescu, O.; Fan, C.; Raichel, B. Largest Red Blue Separating Rectangles Revisited. In Proceedings of the Fall Worksop on Computational Geometry, New York, NY, USA, 27–28 October 2016.
4. Bitner, S.; Cheung, Y.K.; Daescu, O. Minimum Separating Circle for Bichromatic Points in the Plane. In Proceedings of the International Symposium on Voronoi Diagrams in Science and Engineering, Quebec, QC, Canada, 28–30 June 2010; pp. 50–55.
5. Armaselu, B.; Daescu, O. Dynamic Minimum Bichromatic Separating Circle. *Theor. Comput. Sci.* **2019**, *774*, 133–142. [CrossRef]
6. Armaselu, B. Improved Algorithm for Computing the Maximum-volume Bichromatic Separating Box. *arXiv* **2020**, arXiv:2012.128468.
7. Sheikhi, F.; Mohades, A.; de Berg, M.; Mehrabi, A.D. Separability of imprecise points. *Comput. Geom.* **2017**, *61*, 24–37. [CrossRef]
8. Agarwal, B.; Suri, S. Fast algorithms for computing the largest empty rectangle. In Proceedings of the Symposium on Computational Geometry, Waterloo, ON, Canada, 8–10 June 1987; pp. 278–290.
9. Namaad, A.; Lee, D.T.; Hsu, W.L. On the maximum empty rectangle problem. *Discret. Appl. Math.* **1984**, *8*, 267–277. [CrossRef]
10. Mukhopadhyay, A.; Rao, S.V. Computing a Largest Empty Arbitrary Oriented Rectangle. Theory and Implementation. *Int. J. Comput. Geom. Appl.* **2003**, *13*, 257–271. [CrossRef]
11. Chaudhuri, J.; Nandy, S.C.; Das, S. Largest empty rectangle among a point set. *J. Algorithms* **2003**, *46*, 54–78. [CrossRef]
12. Nandy, S.C.; Bhattacharya, B.B.; Ray, S. Efficient algorithms for identifying all maximal isothetic empty rectangles in VLSI layout design. In Proceedings of the International Conference onFoundations of Software Technology and Theoretical Computer Science, Bangalore, India, 17–19 December 1990; pp. 255–269.

13. Nandy, S.C.; Bhattacharya, B.B.; Sinha, A. Location of the largest empty rectangle among arbitrary obstacles. In Proceedings of the Foundations of Software Technology and Theoretical Computer Science, Madras, India, 15–17 December 1994; pp. 159–170.
14. Armaselu, B. Extensions of the Maximum Bichromatic Separating Rectangle Problem. In Proceedings of the Canadian Conference on Computational Geometry, Halifax, NS, Canada, 10–12 August 2021; pp. 237–247.