Maximum Separability by L-shapes

Farnaz Sheikhi
Faculty of Computer Engineering
K. N. Toosi University of Technology
Tehran, Iran
f.sheikhi@kntu.ac.ir

Ali Mohades
Faculty of Mathematics and Computer Science
Amirkabir University of Technology
Tehran, Iran
mohades@aut.ac.ir

***Abstract*—Let B be a set of blue points and R be a set of red points with total size n in the plane. In this paper, we propose a worst-case optimal $O(n^3)$ time algorithm to compute all axis-aligned L-shapes that contain maximum number of blue points without containing any red points. We also study this problem for arbitrarily oriented L-shapes, and present an $O(n^4\alpha(n))$ time algorithm to find these general L-shapes, where $\alpha(n)$ is the inverse of the Ackermann function.**

***Index Terms*—computational geometry, separability, point sets**

I. Introduction

Arising from the facility location applications, a vast area of research in computational geometry has focused on *covering problems* in which given a set P of n points in the plane, the goal is to find the minimum size geometric shape that covers all the points. These problems are broadly studied for different shapes (*covers*) such as a circle [8], rectangle [14], and an L-shape [4]. Importance of studying covering problems has made them more realistic through years of research. In real applications not all points in P are of the same type. There can exist desirable and undesirable points in the input where covering the undesirable points together with the desirable ones is harmful. This discrimination in the input is often modeled by assigning different colors to points: blue to model the desirable points and red to model the undesirable ones. The covering problem is now treated as a *separability problem* in which the goal is to use a predefined geometric shape (*separator*) to cover blue points without covering any red points. Ideally, separating all the blue points from the red points is desired. Let *complete separability* refer to this category of separability problems. Complete separability is studied for separators such as a line [8], circle [9], rectangle [15], and an L-shape [13]. See [11] for a thorough study.

Although complete separability is the most natural type of separability, it is not always possible. This issue and applications in pattern recognition and data analysis [7] have led to the study of *maximum separability* in which the goal is to separate most of the blue points from the red points. Maximum separability has been studied for separators such as a convex polygon [5], circle [2], and rectangle [3], [12]. Considering other types of separators, especially the ones previously studied in the other category of separability problem is left for further research [2].

Thus, given a set R of red points and a set B of blue points with total size n in the plane, in this paper we study maximum separability of B and R by using L-shaped separators. First we study a basic version of this problem where L-shapes are axis-aligned. We give an $O(n^3)$ time algorithm to solve this problem and show that the proposed algorithm is worst-case optimal. We use this algorithm as a basis to design an $O(n^4\alpha(n))$ time algorithm to solve maximum separability of B and R by using arbitrarily oriented L-shapes, where $\alpha(n)$ is the slow-growing inverse of the Ackermann function.

II. Preliminaries

We start by focusing on some definitions and notations. We define an *axis-aligned L-shape* to be an axis-aligned rectangle M that has lost an axis-aligned M' from its top-right corner, where $M' \subsetneq M$. Then an *L-shape with orientation* θ is an axis-aligned L-shape that has been rotated in counterclockwise direction over an angle θ, where $\theta \in [0, 2\pi)$. Given point sets B and R with total size n in the plane, let $\mathcal{P}$ denote the bounding box of $B \cup R$ in the current coordinate frame. Further, for a point p, let x_p and y_p respectively denote the x- and y-coordinate of p. We wish to find L-shapes in $\mathcal{P}$ that contain maximum number of blue points without containing any red points. We call these L-shapes *maximum blue L-shapes* or *MBL*s for short. An MBL can be enlarged to have each side touched by a red point or the boundary of $\mathcal{P}$. Once we design an algorithm to compute MBLs that have a red point on each side, the algorithm can be easily modified to handle the cases where MBLs have some sides coincided with the boundary of $\mathcal{P}$, as discussed in Section III. So for now, we focus on finding MBLs with a red point on each side.

Starting from the top side of a potential MBL or $PMBL$ for short, and traversing its boundary in clockwise order, let $tp, mid-rt, mid-tp, rt, btm$, and lt respectively denote the red point on the top, middle-right, middle-top, right, bottom, and left side of that $PMBL$. See Fig. 1(a). Thus, a trivial solution for computing axis-aligned MBLs is to choose the six red points defining the boundary of a $PMBL$, check whether this $PMBL$ contains a red point inside, count the number of blue points it covers, and repeat the procedure to compute the actual MBLs by comparing the number of blue points in $PMBL$s. This leads to an $O(n^7)$-time solution for this problem. However, next we show how to compute all axis-aligned MBLs in the worst-case optimal $O(n^3)$ time.

978-1-7281-5937-9/20/$31.00 ©2020 IEEE

III. Finding axis-aligned MBLs

To compute MBLs in the axis-aligned case, the main idea is to define a *skeleton* for an L-shape, and then *fatten* this skeleton to get the corresponding $PMBL$s. Details are as follows. Our algorithm has a pre-processing stage where we sort the points in $B \cup R$ according to the x- and y-coordinate. Then we compute the so called skeleton of a $PMBL$. Recall that a $PMBL$ has a red point on each side, where tp is the red point on the top, and rt is the red point on the right side. To define the skeleton of a $PMBL$ we draw two axis-aligned rays, one downward from tp and the other leftward from rt until they meet. The concatenation of the two resultant axis-aligned segments is called the skeleton of the $PMBL$ having tp on the top and rt on the right side. The point where the two segments of the skeleton meet is called the *ankle*. See Fig. 1(a). So, in the body of the algorithm we choose two arbitrary red points, one in the role of tp and the other in the role of rt. Having tp and rt, and consequently the skeleton defined by them, now we describe how to fatten this skeleton to find the corresponding $PMBL$s. That is, to find the red points defining the corresponding $PMBL$s. Consider the axis-aligned rectangle with the upper-right corner (x_{rt}, y_{tp}) and the lower-left corner coincided with the lower-left corner of $\mathcal{P}$. Let $\mathcal{P}'$ denote this rectangle. Drawing axis-aligned rays from tp and rt, we can partition $\mathcal{P}'$ into four rectangular regions: $\mathcal{R}_1$ (the upper-right region), $\mathcal{R}_2$ (the upper-left region), $\mathcal{R}_3$ (the lower-left region), and $\mathcal{R}_4$ (the lower-right region). See Fig. 1(b). By $\mathcal{R}_i$, where $1 \leq i \leq 4$, we mean the open region, and we denote the boundary of $\mathcal{R}_i$ by $\partial\mathcal{R}_i$. We can restrict the search (for finding the remaining red points defining the $PMBL$s with the specified skeleton) to these regions.

Finding $mid-rt$ and $mid-tp$ red points: $\mathcal{R}_1$ is the region we can find $mid-rt$ and $mid-tp$. The point $mid-rt$ is the red point in $\mathcal{R}_1$ with the minimum x-coordinate, and $mid-tp$ is the red point in $\mathcal{R}_1$ with the minimum y-coordinate. See Fig. 1(b). To find these red points we use the segment-dragging algorithm [6], once with the vertical segment of the skeleton, proceeding rightward until it reaches a red point in $\mathcal{R}_1$ (to find $mid-rt$), and the other time with the horizontal segment of the skeleton, proceeding upward until it reaches a red point in $\mathcal{R}_1$ (to find $mid-tp$). If no such points are found, we take (x_{rt}, y_{tp}) as the virtual $mid-rt$ and $mid-tp$. So by an $O(n \log n)$ pre-processing time, we can find $mid-rt$ and $mid-tp$ in $O(\log n)$ time [6].

Finding lt and btm red points: To find lt and btm in the regions $\mathcal{R}_2$ and $\mathcal{R}_4$, we follow a similar segment-dragging approach. To find lt in $\mathcal{R}_2$, we drag the vertical segment of the skeleton leftward until it reaches a red point in $\mathcal{R}_2$ or the boundary of $\mathcal{P}'$. We denote this red point by $far-lt$. Moreover, to find btm in $\mathcal{R}_4$, we drag the horizontal segment of the skeleton downward until it reaches a red point in $\mathcal{R}_4$ or the boundary of $\mathcal{P}'$. We denote this red point by $far-btm$. Having found $far-lt$ and $far-btm$, we shrink $\mathcal{P}'$ to have $far-lt$ on the left and $far-btm$ on the bottom side. This results in shrinking the regions $\mathcal{R}_2$, $\mathcal{R}_3$, and $\mathcal{R}_4$ as well. From now on, by $\mathcal{P}'$ we mean the shrunk $\mathcal{P}'$. Note that $far-lt$ and $far-btm$ are not the only red points that can play the role of lt and btm. This is due to the *maximal* red points in $\mathcal{R}_3$. We say that a red point $r \in \mathcal{R}_3$ is maximal if there does not exist another red point $r^* \in \mathcal{R}_3$ such that $x_{r^*} > x_r$ and $y_{r^*} > y_r$. Connecting the maximal red points in $\mathcal{R}_3$ we get a staircase which we call the *critical-staircase*. See Fig. 1(b). The critical-staircase has $O(n)$ steps, where *extension* of each of these steps defines a $PMBL$ with the specified skeleton. By extension of a step we mean drawing a vertical ray upward from the maximal red point on the vertical segment of the step, and drawing a horizontal ray rightward from the maximal red point on the horizontal segment of the step, until these rays touch $\partial\mathcal{P}'$ (the boundary of $\mathcal{P}'$). Thus, by fattening a specific skeleton we get $O(n)$ corresponding $PMBL$s. What is left is counting the number of blue points in these $PMBL$s. The steps in the critical-staircase can be ordered from the topmost to the bottommost, and so are the corresponding $PMBL$s. Let $PMBL_i$ denote the $PMBL$ that is constructed by extension of the i-th step of the critical-staircase. The first $PMBL$ ($PMBL_1$) has $far-lt$ on the left side and the topmost red point of the critical-staircase on the bottom side while the last $PMBL$ has the bottommost red point of the critical-staircase on the left side and $far-btm$ on the bottom side. See Fig. 1(c). Let $|\mathcal{S}|$ denote the number of blue points in shape $\mathcal{S}$. Next we describe the relation between $|PMBL_i|$ and $|PMBL_{i+1}|$. Drawing vertical rays upward and horizontal rays rightward from the maximal red points on the critical-staircase until they reach $\partial\mathcal{P}'$, we create $O(n)$ vertical and horizontal slabs. Assume that $vslab_i$ and $hslab_i$ respectively denotes the i-th vertical slab from the left and the i-th horizontal slab from the top. We have: $|PMBL_{i+1}| = |PMBL_i| - |vslab_i| + |hslab_{i+1}|$. See Fig. 1(c).

Hence, to count the number of blue points in the $PMBL$s, first we compute $|PMBL_1|$, and then follow an update procedure to compute the number of blue points in the remaining $PMBL$s. We compute $|PMBL_1|$ in $O(n)$ time. To follow the update procedure we use two orthogonal sweeps moving sequentially slab by slab: a vertical sweep line passing through $far-lt$ proceeding rightward, and a horizontal sweep line passing through the topmost maximal red point in $\mathcal{R}_3$ proceeding downward. We use these sweeps to count the number of blue point in the vertical and horizontal slabs, and use these numbers to update $|PMBL_i|$ to get $|PMBL_{i+1}|$, for $1 \leq i \leq n-1$. We continue this procedure until reaching the last $PMBL$. This takes $O(n)$ time per skeleton. Having finished the counting procedure, we achieve the MBLs with the specified skeleton.

It is time to show how to handle the cases where $PMBL$s have the top side or the right side coincided with $\partial\mathcal{P}$. We handle these cases as follows. If $\mathcal{P}$ has no red points inside, then it is the solution itself. Otherwise, from the red points inside $\mathcal{P}$ we draw axis-aligned rays upward and rightward until the rays reach $\partial\mathcal{P}$. This partitions the top side and the right side of $\mathcal{P}$ into $O(n)$ intervals. We consider a virtual red point

in each of these intervals, and handle these cases similarly to the main algorithm described. Thus, we can assume that each $PMBL$ has a red point (either real or virtual) on each side.

Theorem 1. *Let B be a set of blue points and R be a set of red points, with total size n in the plane. Then, we can compute all axis-aligned MBLs in $O(n^3)$ time and $O(n)$ storage.*

Proof: The pre-processing stage of the algorithm takes $O(n \log n)$ time and $O(n)$ storage. Then, in the body of the algorithm we choose two arbitrary red points and compute the corresponding skeleton. This results in $O(n^2)$ skeleton. Fattening each skeleton to achieve the corresponding $PMBL$s and counting the number of blue points in these $PMBL$s by using two orthogonal sweeps take $O(n)$ time. Thus, the total time complexity of the algorithm is $O(n^3)$. During the algorithm we update $|MBL|$ which has been discovered so far such that by the end of the algorithm the actual $|MBL|$ is available. Having $|MBL|$ available, we can report all MBLs in $B \cup R$ in $O(n^3)$ time, using $O(n)$ storage. ∎

The lower bound. We show that there exist point sets that admit $\Omega(n^3)$ MBLs. First we consider two red points in the role of tp and rt. These two red points partition $\mathcal{P}'$ into the regions $\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{R}_3$, and $\mathcal{R}_4$, as described earlier. We put $(n-6)/4$ red points in $\mathcal{R}_1$ on a staircase structure between tp and rt. We place $far-lt$ in $\mathcal{R}_2$ so that it lies above rt and below the bottommost red point on the staircase structure above rt. Similarly, we place $far-btm$ in $\mathcal{R}_4$ so that it lies to the right of tp and to the left of the topmost red point on the staircase structure below tp. Having specified the location of $far-lt$ and $far-btm$, we put $(n-6)/4$ red points on a staircase structure in $\mathcal{R}_3$ (playing the role of the critical-staircase). Finally, we place a single blue point in each step of this critical-staircase as well as in each step of the staircase structure in $\mathcal{R}_1$. See Fig. 2(a). This way, having specified tp and rt, we get $\Theta(n)$ MBLs, where $|MBL|=3$. Further, we have $\Theta(n)$ choices for tp (the red points below tp that lie on the staircase structure in $\mathcal{R}_1$) and also $\Theta(n)$ choices for rt (the red points to the left of rt that lie on the staircase structure in $\mathcal{R}_1$), where for each of these $\Theta(n^2)$ choices we have $\Theta(n)$ MBLs with three blue points inside.

Theorem 2. *Let B be a set of blue points and R be a set of red points, with total size n in the plane. Then, computing all axis-aligned MBLs requires $\Omega(n^3)$ time in the worst-case.*

IV. Finding arbitrarily oriented MBLs

To compute arbitrarily oriented MBLs, first we compute the axis-aligned skeleton and the corresponding $PMBL$s for any pair of red points in the role of (tp, rt), using the algorithm in Section III. With a slight difference, in this section we consider $|PMBL_i| := |\overline{PMBL_i}| + |\mathcal{C}|$, where $1 \le i \le n$, $\mathcal{C}$ is the common region shared among all $PMBL$s defined by this tp and rt, and $\overline{PMBL_i} := PMBL_i \backslash \mathcal{C}$. See Fig. 2(b). This decomposition helps us for a faster update of $|PMBL|$s in Section IV-A. To compute arbitrarily oriented $PMBL$s having tp on the top side and rt on the right side, we use

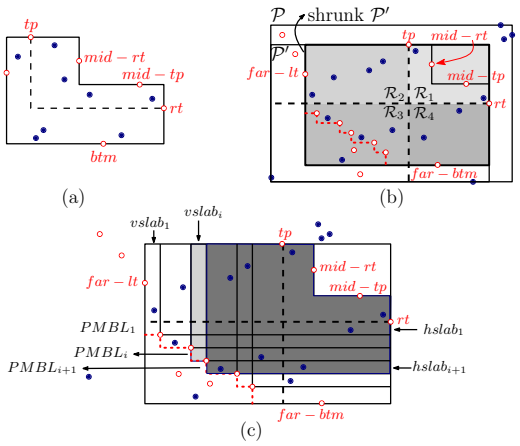

Fig. 1. (a) A $PMBL$ and the corresponding skeleton. The skeleton is shown dashed. (b) Finding the $PMBL$s corresponding to a specific skeleton. (c) Illustrating the relation between $|PMBL_i|$ and $|PMBL_{i+1}|$.

a rotational sweep. We can think of this sweep as rotating the coordinate frame once in counterclockwise and the other time in clockwise direction, and handle the events that arise during the sweep. This is to cover all arbitrarily oriented $PMBL$s with the specified tp and rt. We focus on rotating the coordinate frame in counterclockwise direction. Then handling the rotation in clockwise direction is similar. Thus, we increase θ (the angle that the current positive x-axis makes with the original x-axis) from 0 to 2π, and handle the events (changes in the $PMBL$s) that occur while this increase (rotational sweep). Generally, these events happen when the order of two points changes. Hence, there are $O(n^2)$ events in total and they will be studied in details in Section IV-A.

There are some points to consider before getting to events. As the rotational sweep proceeds we keep the points in $B \cup R$ sorted. To get that, in the pre-processing stage we sort the points according to the original x- and y-coordinate. During the rotational sweep, the order of points needs to be updated. The number of changes in the order of points is $O(n^2)$. Thus, in the pre-processing stage we keep the $O(n^2)$ angles defined by pairs of points sorted in $O(n^2 \log n)$ time and $O(n^2)$ storage. Having this available, during the sweep we can update the order of points in $O(1)$ time. Now let CH_{R-in} be the convex hull of red points in $\mathcal{R}_1$, and CH_{R-left} be the convex hull of red points to the left of $far-lt$ and below tp. We store these hulls in a dynamic convex hull data structure that maintains an explicit representation of the hull [10]. This takes $O(n \log^2 n)$ time. During the rotation we also keep these convex hulls up to date. We define two variables: θ_{max} (storing the orientation of MBL) and $|\mathcal{C}_{max}|$ (storing the maximum number of blue points in $\mathcal{C}$ while rotation). At the start of the sweep, we set $\theta_{max} := 0$ and $|\mathcal{C}_{max}| := |\mathcal{C}|$. These variables get updated during the algorithm.

We have the $O(n^2)$ sorted angles defined by pairs of points in the pre-processing stage. Then, we traverse these angles in order and detect the events corresponding to them. Let θ_k be the $k-$th angle in the list of sorted angles. We detect the type

of θ_k to handle as follows.

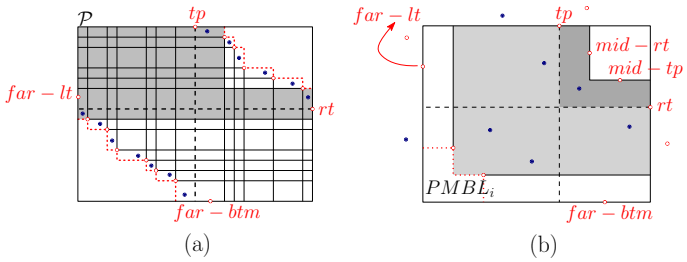

Fig. 2. (a) The lower bound on the number of axis-aligned MBLs. The shaded area shows a sample axis-aligned MBL. (b) Decomposing $PMBL_i$: $\overline{PMBL_i}$ is the light gray while $\mathcal{C}$ is the dark gray region.

A. Events

TP-touching event: This event occurs when the line through tp touches a point while rotation. We can recognize this event in $O(1)$ time, and there are $O(n)$ number of them. This event is of two types: TPH and TPV.

TPH: This type of TP-touching event occurs when the horizontal line through tp touches a point while rotation. Let this point be a red one, and r denote it. If $r \in \partial\mathcal{R}_1$ then we update CH_{R-in} in $O(\log^2(n))$ time [10]. Further, if $r \in \partial\mathcal{R}_1$ and r is to the left of $mid-rt$ then r will play the role of the new $mid-rt$. So we update $mid-rt$, $\mathcal{C}$ and consequently $|\mathcal{C}|$ in $O(n)$ time. See Fig. 3(a). If r is to the left of tp, then we update CH_{R-left} [10]. In this case if r is the same as $far-lt$ (as in Fig. 3(b)) then it leaves $\mathcal{R}_2$, and a new $far-lt$ should be computed. So we update $far-lt$ and consequently the whole structure ($PMBL$s and the number of blue points in them). This takes $O(n)$ time. On the other hand, let the horizontal line through tp touches a blue point, and b denote this blue point. If $b \in \partial\mathcal{R}_1$ and b is to the left $mid-rt$ (as in Fig. 3(c)) then b enters $\mathcal{C}$. Thus, we update $|\mathcal{C}|$. If $|\mathcal{C}| > |\mathcal{C}_{max}|$ then we set $|\mathcal{C}_{max}| = |\mathcal{C}|$ and $\theta_{max} = \theta_k$. This takes $O(1)$ time. If $b \in \partial\mathcal{R}_2$ then b leaves $\mathcal{R}_2$ and some of $PMBL$s. See Fig. 3(d). To find these $PMBL$s, by a binary search on the maximal red points in $\mathcal{R}_3$ we find the vertical slab which b falls in, in $O(\log n)$ time. Let $vslab_i$ denote this slab. Then moving top-down we need to update $|PMBL_1|$ until $|PMBL_i|$. To this aim we use a similar approach to Section III. This way the update takes $O(n)$ time.

TPV: This event occurs when the vertical line through tp touches a point while rotation. Let this point be a red point r. If r is the same as $far-btm$ (as in Fig. 3(e)) then we update $far-btm$ and the whole structure in $O(n)$ time. If r is the same as $mid-rt$ (as in Fig. 3(f)), then CH_{R-in} and CH_{R-left} get updated. Moreover, we update $mid-rt$, recompute $\mathcal{C}$, $|\mathcal{C}|$ and the whole structure in $O(n)$ time. If $|\mathcal{C}| > |\mathcal{C}_{max}|$ then we set $|\mathcal{C}_{max}| = |\mathcal{C}|$ and $\theta_{max} = \theta_k$. On the other hand, let the vertical line through tp touches a blue point, and b denote it. If $b \in \partial\mathcal{R}_1$ (as in Fig. 3(g)), b leaves $\mathcal{C}$. So $|\mathcal{C}|$ gets updated in $O(1)$ time.

Lemma 3. *The number of TP-touching events is $O(n)$. Each TP-touching event (either TPH or TPV) can be recognized in $O(1)$ time, and handled in $O(n)$ time.*

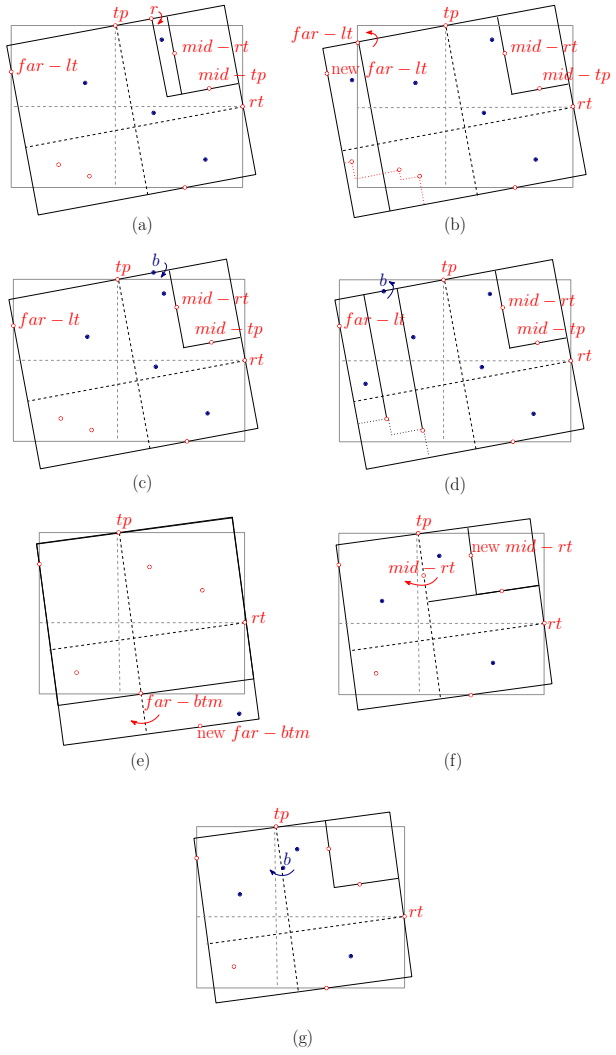

Fig. 3. Illustrating TP-touching events.

RT-touching event: This event occurs when the line through rt touches a point while rotation. We can recognize this event in $O(1)$ time, and there are $O(n)$ number of them. It is of two types: RTH and RTV.

RTH: This type of RT-touching event occurs when the horizontal line through rt touches a point while rotation. Let this point be a red one, and r denote it. If r is the same as the topmost maximal red point in $\mathcal{R}_3$ then it will play the role of the new $far-lt$. So we update $far-lt$ and consequently the whole structure in $O(n)$ time. See Fig. 4(a). On the other hand, let the horizontal line through rt touches a blue point, and b denote it. If b is in $hslab_1$ and to the right of tp (as in Fig. 4(b)), then it enters $\mathcal{C}$. Thus, $|\mathcal{C}|$ gets updated in $O(1)$ time. If $|\mathcal{C}| > |\mathcal{C}_{max}|$ then we set $|\mathcal{C}_{max}| = |\mathcal{C}|$ and $\theta_{max} = \theta_k$.

RTV: This type of RT-touching event occurs when the vertical line through rt touches a point while rotation. Let this point be a red one, and r denote it. If $r \in \partial\mathcal{R}_1$ then r leaves

$\mathcal{R}_1$. So we update CH_{R-in} [10]. Further, if r is the same as $mid-tp$ (as in Fig. 4(c)), then we need to update $mid-tp$ and $|\mathcal{C}|$ as well. This takes $O(n)$ time. If $|\mathcal{C}| > |\mathcal{C}_{max}|$ then we set $|\mathcal{C}_{max}| = |\mathcal{C}|$ and $\theta_{max} = \theta_k$. If $r \in \partial\mathcal{R}_4$ then r enters $\mathcal{R}_4$ as the new $far-btm$. See Fig. 4(d). So we update $far-btm$ and consequently the whole structure in $O(n)$ time. On the other hand, let the vertical line through rt touches a blue point, and b denote it. If $b \in \partial\mathcal{R}_1$ and b is below $mid-tp$ then b leaves $\mathcal{C}$. See Fig. 4(e). Thus, we update $|\mathcal{C}|$. Otherwise, if $b \in \partial\mathcal{R}_4$ then b enters $\mathcal{R}_4$ and some of $PMBL$s. See Fig. 4(f). To find these $PMBL$s, by using a binary search on the maximal red points in $\mathcal{R}_3$ we find the horizontal slab that b falls in. Let $hslab_i$ be this slab. Then we update the number of blue points in $PMBL_j$, where $i \leq j \leq n$, in $O(n)$ time.

Lemma 4. *The number of RT-touching events is $O(n)$. Each RT-touching event (either RTH or RTV) can be recognized in $O(1)$ time, and handled in $O(n)$ time.*

$MID-RT$-touching event: This event occurs when the vertical line through $mid-rt$ touches a specific point while rotation. There are two types of $MID-RT$-touching events: $MIDR$ and $MIDB$.

$MIDR$: This type of event occurs when the vertical line through $mid-rt$ touches a red point which is the counter-clockwise neighbor of $mid-rt$ on CH_{R-in}. See Fig. 5(a). Let r denote this red point. $MIDR$ event makes r to be the new $mid-rt$. Since $mid-rt$ and CH_{R-in} are available, this event can be recognized and handled in $O(1)$ time. Note that this event corresponds to changes in $mid-rt$. That is, changes in the red point in $\mathcal{R}_1$ with the minimum x-coordinate in the current coordinate frame. So for each red point $r \in \mathcal{R}_1$, we define the function $f_r(\theta) = x_r(\theta)$, where $x_r(\theta)$ is the x-coordinate of r in the coordinate frame that is rotated by the angle θ in counterclockwise direction. For the rest of red points this function is undefined. Within the angular interval $[0, 2\pi)$ the number of times that a red point r may enter or leave $\mathcal{R}_1$, and consequently the number of pieces of $f_r(\theta)$ is $O(1)$. So, over all red points we have $O(n)$ pieces of functions. The red point in $\mathcal{R}_1$ with the minimum x-coordinate is achieved by the lower envelope of these functions. Since any two pieces of these functions intersect in at most one point, the complexity of their lower envelope is $O(n\alpha(n))$, where $\alpha(n)$ is the inverse of the Ackermann function [1]. Thus, the number of $MIDR$ events is $O(n\alpha(n))$, and each can be recognized and handled in $O(1)$ time.

$MIDB$: This type of $MID-RT$-touching event occurs when the vertical line through $mid-rt$ touches a blue point in $\mathcal{R}_1$. Let b be such a blue point. If b is above $mid-rt$ and below tp then b leaves $\mathcal{C}$. See Fig. 5(b). Otherwise, if b is below $mid-rt$ and above $mid-tp$ then b enters $\mathcal{C}$. See Fig. 5(c). Hence, according to each case we update $|\mathcal{C}|$ in $O(1)$ time. If $|\mathcal{C}| > |\mathcal{C}_{max}|$ then we set $|\mathcal{C}_{max}| = |\mathcal{C}|$ and $\theta_{max} = \theta_k$. The number of $MIDB$ events is $O(n^2)$, and each of them can be recognized and handled in $O(1)$ time.

Lemma 5. *Among $MID-RT$-touching events, the number of $MIDR$ events is $O(n\alpha(n))$ while the number of $MIDB$ events is $O(n^2)$. Each of these events can be recognized and handled in $O(1)$ time.*

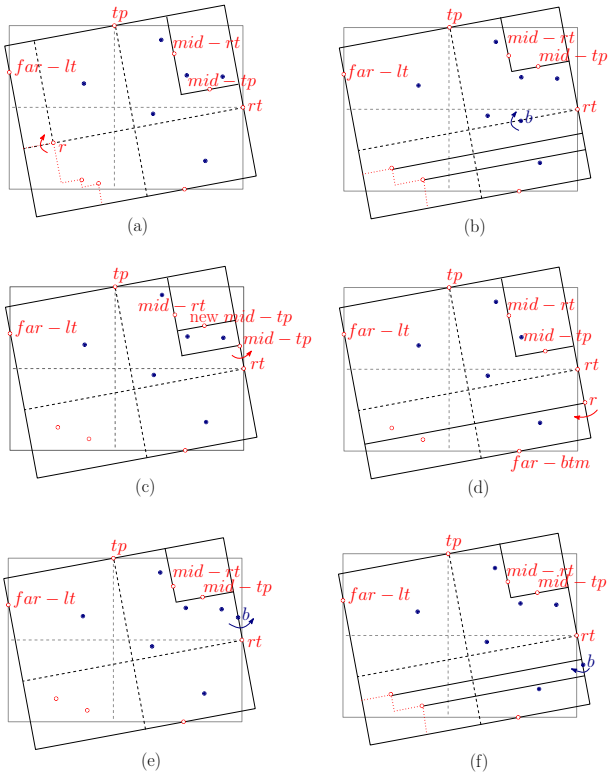

Fig. 4. Illustrating RT-touching events.

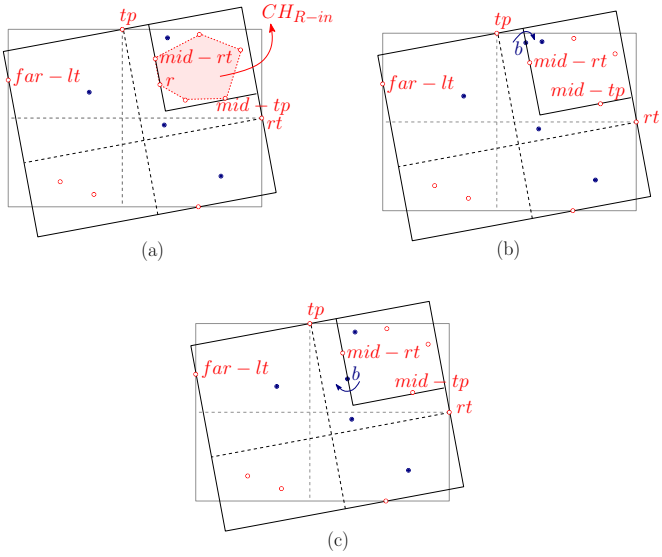

Fig. 5. Illustrating $MID-RT$-touching events.

$MID-TP$-touching event: This event occurs when the horizontal line through $MID-TP$ touches a point while rotation. It can be analyzed similar to $MID-RT$-touching event, and hence further details are omitted.

$FAR-LT$-touching event: This event occurs when the vertical line through $far-lt$ touches a specific point (described in the following) while rotation. This event is of two types: $FAR-LT_{\text{red}}$ and $FAR-LT_{\text{blue}}$.

$FAR-LT_{\text{red}}$: This type of event occurs when the vertical line through $far-lt$ touches a red point which is the counterclockwise neighbor of $far-lt$ on CH_{R-left}. Let r be this red point. Since $far-lt$ and CH_{R-left} are available, a $FAR-LT_{\text{red}}$ event can be recognized in $O(1)$ time. This event makes r to become the new $far-lt$. See Fig. 6(a). This update takes $O(1)$ time. The number of $FAR-LT_{\text{red}}$ events is the number of times that $far-lt$ gets collinear with its counterclockwise neighbor on CH_{R-left}. It can be interpreted as the number of vertices on CH_{R-left} that is traversed while rotation. This traverse is only done from right to left on CH_{R-left}, and it never gets back. Although CH_{R-left} is updated at TP-touching events, the total number of $FAR-LT_{\text{red}}$ events is still $O(n)$.

$FAR-LT_{\text{blue}}$: This type occurs when the vertical line through $far-lt$ touches a blue point b while rotation. If b is above $far-lt$ and below tp then b enters $\mathcal{R}_2$, $vslab_1$ and consequently $PMBL_1$. See Fig. 6(b). Otherwise, if b is below $far-lt$ and above the topmost maximal red point in $\mathcal{R}_3$ then b leaves $\mathcal{R}_2$, $vslab_1$ and consequently $PMBL_1$. See Fig. 6(c). So according to each case we update $|PMBL_1|$. A $FAR-LT_{\text{blue}}$ event can be recognized and handled in $O(1)$ time. The number of $FAR-LT_{\text{blue}}$ events is $O(n^2)$.

Lemma 6. *Among $FAR-LT$-touching events, the number of $FAR-LT_{\text{red}}$ events is $O(n)$ while the number of $FAR-LT_{\text{blue}}$ events is $O(n^2)$. Each of these events can be recognized and handled in $O(1)$ time.*

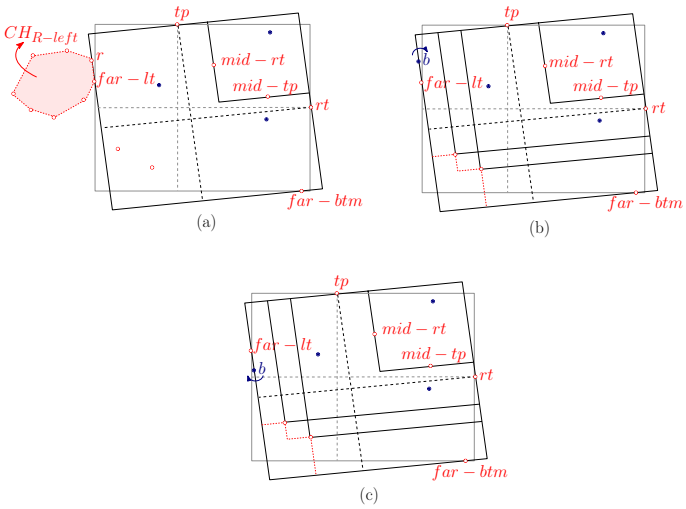

Fig. 6. Illustrating $FAR-LT$-touching events.

STEP-touching event: This event occurs when a step through the maximal red points in $\mathcal{R}_3$, including $far-lt$ and $far-btm$, touches a point while rotation. This event is of two types: STEP_{red} and $\text{STEP}_{\text{blue}}$.

STEP_{red}: This type of event occurs when a segment through a step touches a red point while rotation. Let r be this red point. According to whether the vertical or the horizontal segment of this step touches r, a STEP_{red} event makes r to be respectively removed from or added to the maximal red points. Hence, the balanced binary search tree that keeps the maximal red points gets updated. Further, $PMBL$s get re-numbered in $O(n)$ time. Moreover, if the horizontal segment through $far-btm$ touches r, then r becomes the new $far-btm$. Thus, a STEP_{red} event can be handled in $O(n)$ time. Similar to $in-events$ and $out-events$ in [4], the number of STEP_{red} events is $O(n)$. These events can be pre-computed in $O(n^2)$ time [4]. Once they are computed, they can be recognized in $O(1)$ and handled in $O(n)$ time.

Lemma 7. *The number of STEP_{red} events is $O(n)$. They can be computed in $O(n^2)$ time. Each of these events can be recognized in O(1) time, and handled in $O(n)$ time.*

$\text{STEP}_{\text{blue}}$: This type occurs when a segment through a step touches a blue point b while rotation. According to whether the vertical or the horizontal segment of this step touches b, a $\text{STEP}_{\text{blue}}$ event makes b to be respectively removed from or added to the corresponding $PMBL$. Hence, the number of blue points in the corresponding $PMBL$ gets updated in $O(1)$ time. The number of $\text{STEP}_{\text{blue}}$ events is $O(n^2)$.

Lemma 8. *The number of $\text{STEP}_{\text{blue}}$ events is $O(n^2)$. Each one can be recognized and handled in $O(1)$ time.*

MAXIMAL-touching event: This event occurs when the line through a maximal red point in $\mathcal{R}_3$ touches a blue point while rotation. Having numbered the maximal red points in $\mathcal{R}_3$ top-down, let r_i be the corresponding maximal red point and b be the corresponding blue point. If the vertical line through r_i touches b, and b is in $vslab_i$, then b leaves $vslab_i$ and enters $vslab_{i+1}$, and consequently $PMBL_{i+1}$. Thus, $|vslab_i|$, $|vslab_{i+1}|$, and $|PMBL_{i+1}|$ get updated. See Fig. 7(a). On the other hand, if the horizontal line through r_i touches b, and b is in $hslab_i$, then b leaves $hslab_i$ and enters $hslab_{i+1}$, and consequently $PMBL_{i+1}$. Thus, $|hslab_i|$, $|hslab_{i+1}|$, $|PMBL_i|$, and $|PMBL_{i+1}|$ get updated. See Fig. 7(b). The update takes $O(1)$ time. Since maximal red points are labeled, recognizing and handling MAXIMAL-touching events take $O(1)$ time, and there are $O(n^2)$ number of these events.

Lemma 9. *The number of MAXIMAL-touching events is $O(n^2)$. Each can be recognized and handled in $O(1)$ time.*

B. Algorithm

To find all arbitrarily oriented MBLs, in the pre-processing stage we compute all $O(n^2)$ angles defined by pairs of points, and keep them sorted. Then we sort the points in $B \cup R$ according to the initial x- and y-coordinate. We set $\theta_{max} := 0$ and $|\mathcal{C}_{max}| := |\mathcal{C}|$, and throughout the algorithm we update these variables as well as the sorted list of points. Thus, the pre-processing stage takes $O(n^2 \log n)$ time and $O(n^2)$ storage. Having completed the pre-processing stage, now for

any pair of red points in the role of (tp, rt) we compute the corresponding axis-aligned $PMBL$s and count the number of blue points in them in $O(n)$ time as in Section III. We store $far - btm$, $far - lt$ and the maximal red points in $\mathcal{R}_3$ in a balanced binary search tree. Now we compute STEP_{red} events in $O(n^2)$ time as in Lemma 7. Next for the pair (tp, rt) we start the rotational sweep. We traverse the $O(n^2)$ angles defined by pairs of points in order, and detect the events that arise while this sweep. At each of these angles we update the order of points in $O(1)$ time. Then we recognize the events in $O(1)$ time as described in Section IV-A, and handle them according to their types. Let θ_k be the current angle to handle. If θ_k is of types TP-touching, RT-touching, $MIDR$, $FAR - LT_{\text{red}}$, or STEP_{red} events, then for θ_{max}, where $\theta_{k-1} < \theta_{max} < \theta_k$, we update $PMBL_j$, where $1 \leq j \leq n$, using the relation $|PMBL_j| := |\overline{PMBL_j}| + |\mathcal{C}_{max}|$. Next we set $|\mathcal{C}_{max}| = |\mathcal{C}|$ and $\theta_{max} = \theta_k$. This takes $O(n)$ time, and there are $O(n\alpha(n))$ events of those types as in Lemmas 3-7. The rest of the $O(n^2)$ angles and events can be handled totally in $O(n^2)$ time as in Lemmas 5, 6, 8, and 9. So by an $O(n^2 \log n)$ pre-processing time and $O(n^2)$ storage, handling each red pair (tp, rt) takes $O(n^2\alpha(n))$ time.

Theorem 10. *Given bichromatic point sets with total size n in the plane, we can compute all arbitrarily oriented MBLs in $O(n^4\alpha(n))$ time and $O(n^2)$ storage.*

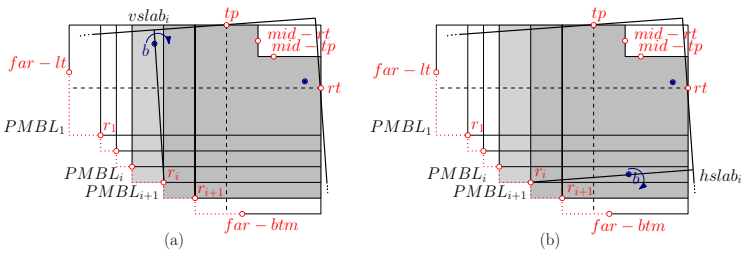

Fig. 7. Illustrating MAXIMAL-touching events.

V. Conclusion

Given a set B of blue points and a set R of red points with total size n in the plane, in this paper we have studied maximum separability of B and R by using L-shaped separators. We have proposed a worst-case optimal $O(n^3)$ time algorithm to compute all axis-aligned L-shapes that contain maximum number of blue points without containing any red points. We have also studied this problem for arbitrarily oriented L-shaped separators, and presented an $O(n^4\alpha(n))$ time algorithm to find these general L-shapes. L-shaped separators are the first non-convex separators studied in the maximum separability problem. Considering other non-convex separators in separability problems is left for further research.

References

[1] P.K. Agarwal and M. Sharir. Davenport–schinzel sequences and their geometric applications. Technical report, Cambridge University Press, 1995.

[2] B. Aronov and S. Har-Peled. On approximating the depth and related problems. In *Proc. 16th Ann. ACM-SIAM Symp. Discr. Alg.*, SODA '05, pages 886–894, 2005.

[3] J. Backer and J.M. Keil. The mono- and bichromatic empty rectangle and square problems in all dimensions. In *LATIN 2010: Theoretical Informatics*, volume 6034 of *LNCS*, pages 14–25. Springer Berlin Heidelberg, 2010.

[4] S.W. Bae, C. Lee, H-K. Ahn, S. Choi, and K-Y. Chwa. Maintaining extremal points and its applications to deciding optimal orientations. In *ISAAC*, pages 788–799, 2007.

[5] C. Bautista-Santiago, J.M. Díaz-Báñez, D. Lara, P. Pérez-Lantero, J. Urrutia, and I. Ventura. Computing optimal islands. *Oper. Res. Lett.*, 39(4):246 – 251, 2011.

[6] B. Chazelle. An algorithm for segment-dragging and its implementation. *Algorithmica*, 3:205–221, 1988.

[7] J. Eckstein, P. Hammer, Y. Liu, M. Nediak, and B. Simeone. The maximum box problem and its application to data analysis. *Comput. Optim. Appl.*, 23(3):285–298, 2002.

[8] N. Megiddo. Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems. *SIAM J. on Comput.*, 12(4):759–776, 1983.

[9] J. O'Rourke, S.R. Kosaraju, and N. Megiddo. Computing circular separability. *Discr. Comput. Geom.*, 1:105–113, 1986.

[10] M.H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Sys. Sci.*, 23(2):166 – 204, 1981.

[11] C. Seara. *On Geometric Separability*. Ph.D. Thesis, Universidad Politécnica De Catalunya, 2002.

[12] F. Sheikhi and A. Mohades. Planar maximum-box problem revisited. *Theor. Comput. Sci.*, 729:57 – 67, 2018.

[13] F. Sheikhi, A. Mohades, M. de Berg, and M. Davoodi Monfared. Separating bichromatic point sets by L-shapes. *Comput. Geom.*, 48(9):673 – 687, 2015.

[14] G. Toussaint. Solving geometric problems with the rotating calipers. *Proc. IEEE MELECON*, pages A10.02/1–4, 1983.

[15] M. van Kreveld, T. van Lankveld, and R. Veltkamp. Identifying well-covered minimal bounding rectangles in 2D point data. In *EuroCG*, pages 277–280, 2009.