The Orthogonal Two-Line Center Problem*

Taehoon Ahn† Sang Won Bae‡ Sang Duk Yoon§

Abstract

Given a set of n points in the plane, the *two-line center problem* asks to find two lines that minimize the maximum distance from each point to its closer line. The best known algorithm for the problem takes $O(n^2 \log^2 n)$ time, presented by Jaromczyk and Kowaluk in 1995. In this paper, we consider the *orthogonal* two-line center problem, a constrained version of the two-line center problem in which two resulting lines should be orthogonal. We present an $O(n^2 \log n)$ time algorithm using only $O(n)$ space.

1 Introduction

In the *two-line center problem*, we are given a set P of n points in $\mathbb{R}^2$ and want to find two lines that minimize the maximum distance to the closer line from each $p \in P$. The first subcubic $O(n^2 \log^5 n)$-time algorithm for the problem was presented in 1991 by Agarwal and Sharir [3]. Later in 1995, Jaromczyk and Kowaluk [15] improved it to $O(n^2 \log^2 n)$ time, and it still remains the best known algorithm until now. See also Katz and Sharir [16] and Glozman et al. [14] for other approaches to the problem.

Whereas no progress on the problem has been made for the last three decades, there has been effort to achieve better running times for variants of two-line center problem with additional constraints. Bae [8] presented an algorithm that runs in $O(n^2)$ time for the *parallel* two-line center problem, in which two resulting lines should be parallel. Very recently, Ahn and Bae [5] studied orientation-constrained versions of the two-line center problem, and presented efficient algorithms with running times: $O(n \log n)$ time when both resulting lines should have fixed orientations, $O(n \log^3 n)$ time when one of them should have a fixed orientation, and $O(n^2 \alpha(n) \log^2 n)$ time when the angle between two resulting lines is fixed, where $\alpha(n)$ denotes the inverse Ackermann function.

Along this line of research, in this paper, we study the *orthogonal* two-line center problem, in which we consider an additional constraint that the two resulting lines should be orthogonal. Our main contribution is a new algorithm for the orthogonal two-line center problem, which takes $O(n^2 \log n)$ time and $O(n)$ space. Note that the orthogonal two-line center problem is a special case of the *fixed-angle* two-line center problem, described above as the third variant of Ahn and Bae [5], whose algorithm makes use of the arrangement of n dual lines via the point-to-line duality and therefore requires $O(n^2)$ space. Thus, our algorithm improves the previous one both in time and space for the orthogonal case.

Related work. Generalizing two-line center problem, the *k-line center problem* is to find k lines that optimally cover a given set P of points in the plane. Notice that the k-line center problem is equivalent to the problem of finding k *strips* of minimum width whose union encloses input points P. The 1-line center problem is known as the *width* problem, and it can be solved in $O(n)$ time after computing the convex hull of P [18]. In $\mathbb{R}^3$ or higher dimensions, Agarwal and Sharir [4] showed that the width of n points in $\mathbb{R}^3$ can be computed in $O(n^{3/2+\epsilon})$ expected time, and Chan [10] discussed an $O(n^{\lceil d/2 \rceil})$-time algorithm in $\mathbb{R}^d$ for $d \geq 4$. For the two-line center problem in $\mathbb{R}^2$, the exact algorithms are known as mentioned earlier, and when the points are given in $\mathbb{R}^d$ for $d \geq 3$, an efficient approximation algorithm for two-line center is presented by Agarwal et al. [1]. The general k-line center problem is known to be NP-hard when k is a part of input [17] even in $\mathbb{R}^2$, while an efficient approximation algorithm is known [2].

In addition to the previous research [5, 8] introduced above, more results on variants of the k-line center problem and their extensions to higher dimensions have recently been studied. Das et al. [12] presented an approximation algorithm for the *axis-aligned* k-line center problem in which resulting lines should be either horizontal or vertical. Chung et al. [11] presented algorithms that compute k parallel strips of best separability, enclosing a given planar set of points. Ahn et al. [6] presented algorithms computing two parallel slabs enclosing points in $\mathbb{R}^d$ for $d \geq 3$.

*T. Ahn was supported by the National Institute for Mathematical Sciences (NIMS) Grant funded by the Korean government(MSIT) (No. NIMS-B25910000). S.W. Bae was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. RS-2023-00251168).

†National Institute for Mathematical Sciences, Daejeon, Korea. taehoon@nims.re.kr

‡Division of AI Computer Science and Engineering, Kyonggi University, Suwon, Korea. swbae@kgu.ac.kr

§Department of Service and Design Engineering, Sungshin Women's University, Seoul, Korea. sangduk.yoon@sungshin.ac.kr

Due to space limit, proofs are omitted.

2 Preliminaries

Let P be a given set of n points in the plane $\mathbb{R}^2$. We assume that the points in P are in general position; that is, no three points of P lie on a common line. We consider the standard Cartesian coordinate system of $\mathbb{R}^2$ with the *horizontal* x-axis and the *vertical* y-axis. For each point $p \in \mathbb{R}^2$, we use $x(p)$ and $y(p)$ to denote its x- and y-coordinates, respectively.

We say that the *orientation* of a line is $\theta \in [0, \pi)$ (modulo π) if θ is the counterclockwise angle from the x-axis to the line. A line of orientation θ may have a *direction* $\phi \in [0, 2\pi)$, where $\phi = \theta$ or $\phi = \theta + \pi$. For a point $p \in \mathbb{R}^2$, we use $\vec{\ell}_\phi(p)$ to denote the line in direction ϕ that goes through p. A half-line (or a ray) starting at p in direction ϕ is the subset of $\vec{\ell}_\phi(p)$ consisting of all points on the line but those prior to p in direction ϕ.

A *strip* σ is the closed region bounded by two parallel lines, and its orientation is determined as that of its bounding lines. The *width* of strip σ, denoted by $\mathrm{w}(\sigma)$, is the distance between its two bounding lines. For any subset $Q \subseteq P$, we denote by $\sigma_\theta(Q)$ the minimum-width strip of orientation θ that encloses Q.

Note that the orthogonal two-line center problem is equivalent to the problem of finding a pair of orthogonal strips that encloses P such that the width of the wider strip is minimized. For a pair of orthogonal strips σ_1 and σ_2, we call the union $C = \sigma_1 \cup \sigma_2$ of the strips a *cross*. The *orientation* of cross C is meant to be θ for $\theta \in [0, \pi/2)$ if the orientations of σ_1 and σ_2 are θ and $\theta + \pi/2$, respectively. The *width* of cross C is defined by $\mathrm{w}(C) = \max\{\mathrm{w}(\sigma_1), \mathrm{w}(\sigma_2)\}$. The four bounding lines of σ_1 and of σ_2 are also called bounding lines of C.

Consider any cross C of orientation θ. The complement $\mathbb{R}^2 \setminus C$ consists of four quadrants bounded by half-lines of orientation θ or $\theta + \pi/2$. We denote each of the four quadrants by $Q_i(C)$ for $i \in \{0, 1, 2, 3\}$ and two bounding half-lines of $Q_i(C)$ by $h_{2i}(C)$ and $h_{2i+1}(C)$ in direction $\theta + i\pi/2$ and $\theta + (i+1)\pi/2$, respectively. We simply write Q_i or h_j for indices $0 \le i \le 3$ and $0 \le j \le 7$ instead of $Q_i(C)$ or $h_j(C)$, if it is understood from the context. See Figure 1(a).

3 Data structures

In this section, we describe data structures which will be maintained in our algorithm. We call a cross $C = \sigma_1 \cup \sigma_2$ of orientation θ *minimal* if $\sigma_1 = \sigma_\theta(P \setminus \sigma_2)$ and $\sigma_2 = \sigma_{\theta+\pi/2}(P \setminus \sigma_1)$. If C is minimal, then each bounding line of C contains a point of P unless $P \subset \sigma_1$ or $P \subset \sigma_2$. It is obvious that there exists a minimum-width cross enclosing P that is minimal.

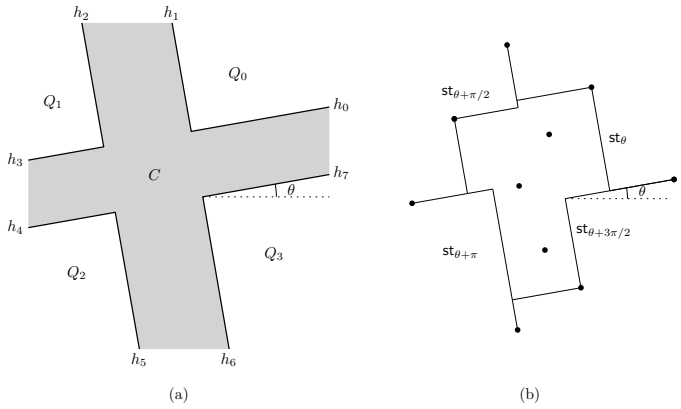
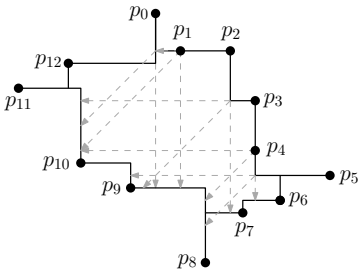


Figure 1: (a) A cross C of orientation θ and its four quadrants Q_0, Q_1, Q_2, and Q_3. Each quadrant Q_i is bounded by two half-lines h_{2i} and h_{2i+1}. (b) Illustration of OH_θ. It consists of four staircases: st_θ, $\mathsf{st}_{\theta+\pi/2}$, $\mathsf{st}_{\theta+\pi}$, and $\mathsf{st}_{\theta+3\pi/2}$.

Let K_p for $p \in \mathbb{R}^2$ be the quadrant with apex p bounded by two rays emanating from p, one horizontally to the left and the other vertically downwards. Consider the union of K_p over all $p \in P$, and the polygonal chain from the topmost point of P to the rightmost one of P along the boundary of the union of quadrants. This rectilinear chain, consisting only of horizontal and vertical segments, is called the *staircase* of P. Note that the staircase can be degenerated to a single point $p \in P$, when $x(p) > x(q)$ and $y(p) > y(q)$ for all other points q in P. Let st_θ for $\theta \in [0, 2\pi)$ denote the staircase of P with the axes rotated by θ in the counterclockwise direction.

For any $\theta \in [0, 2\pi)$, consider the staircase st_θ in orientation θ and let $p_0, \ldots, p_k \in \mathsf{st}_\theta$ be the points of P lying on st_θ in this order. Note that p_0 is the topmost point of P and p_k is the rightmost point of P with the axes rotated by θ. A portion of rectilinear chain st_θ between two consecutive points p_{i-1} and p_i is called a *step*. In general, the i-th step between p_{i-1} and p_i consists of the vertical segment from p_{i-1} down to the point v_i with coordinates $x(v_i) = x(p_{i-1})$ and $y(v_i) = y(p_i)$ and the horizontal segment between v_i and p_i. Note that such a step can be degenerated to a segment, either horizontal or vertical, when p_{i-1} and p_i have the same x- or y-coordinate. We call v_i the *vertex* of the i-th step.

For each $\theta \in [0, \pi/2)$, the four staircases st_θ, $\mathsf{st}_{\theta+\pi/2}$, $\mathsf{st}_{\theta+\pi}$, and $\mathsf{st}_{\theta+3\pi/2}$ form the well-known *orthogonal convex hull* of P with the axes rotated by θ, denoted by OH_θ, and the set of points $p \in P$ lying on the four staircases is called the *maxima* of P in orientation θ. See Figure 1(b). By a slight abuse of notation, the staircase st_θ is also considered as a sequence of points $p \in P$ lying on it, and we also mean by OH_θ the union of the four staircases $\mathsf{st}_{\theta+i\pi/2}$ for $i \in \{0, 1, 2, 3\}$.



Step	p_u^1	p_u^2	p_u^3
p_0p_1	$p_{12}p_0$	$p_{10}p_{11}$	–
p_1p_2	$p_{12}p_0$	$p_{10}p_{11}$	–
p_2p_3	–	p_8p_9	p_7p_8
p_3p_4	–	p_8p_9	p_6p_7
p_4p_5	–	p_8p_9	p_6p_7

Figure 2: The orthogonal convex hull of the 13 points $p_0, ..., p_{12}$. For each step p_ip_{i+1} of st_0 where $\widehat{u}$ is p_ip_{i+1}, the gray arrows indicate the targets of p_u^1, p_u^2, and p_u^3. The information we maintain for each step of st_0 is summarized in the table on the right.

The following observation is immediate.

Lemma 1 *For a cross C of orientation θ that encloses P, any point of P lying on the boundary of $Q_i(C)$ lies on $\mathsf{st}_{\theta+i\pi/2}$ for $i \in \{0,1,2,3\}$.*

This suggests a natural approach to tackle the problem by maintaining OH_θ while θ increases from 0 to $\pi/2$.

Description of data structures. Now, we introduce our data structures that maintain OH_θ over $\theta \in [0, \pi/2)$ and also that supports a specific type of queries for our later purpose. More precisely, let $\mathcal{T}_0, \ldots, \mathcal{T}_3$ be four balanced binary search trees such that $\mathcal{T}_i$ stores the steps of $\mathsf{st}_{\theta+i\pi/2}$ at its leaf nodes in the order along the staircase $\mathsf{st}_{\theta+i\pi/2}$. We add internal pointers between two consecutive leaf nodes so that we can visit the leaf node preceding or succeeding a given leaf node of $\mathcal{T}_i$ in $O(1)$ time. We denote by $\widehat{u}$ the step stored at a leaf node u.

Additionally, we maintain three external pointers p_u^j for $j \in \{1,2,3\}$ to a leaf node of $\mathcal{T}_{i+j}$ (the index of the trees is taken by modulo 4). Precisely, suppose the current $\mathcal{T}_i$'s represent OH_θ for $\theta \in [0, \pi/2)$. Let u be a leaf node of $\mathcal{T}_i$ and v be the vertex of the step $\widehat{u}$ stored at u. We then store three external pointers at node u.

1. p_u^1 points to the leaf node u_1 of $\mathcal{T}_{i+1}$ such that the ray from the vertex v of $\widehat{u}$ in direction $\theta + i\pi/2 + \pi$ hits $\widehat{u_1}$. If there is no such step (and no such leaf node u_1), then p_u^1 points to null.
2. p_u^2 points to the leaf node u_2 of $\mathcal{T}_{i+2}$ such that the ray from v in direction $\theta + i\pi/2 + 5\pi/4$ hits $\widehat{u_2}$. If there is no such step (and no such leaf node u_2), then p_u^2 points to null.
3. p_u^3 points to the leaf node u_3 of $\mathcal{T}_{i+3}$ such that the ray from v in direction $\theta + i\pi/2 + 3\pi/2$ hits $\widehat{u_3}$. If there is no such step (and no such leaf node u_3), then p_u^3 points to null.

Roughly speaking, these external pointers enable us in $O(1)$ time to shoot a ray of a specific direction from the vertex of a step of one staircase to the others at the current orientation θ. See Figure 2 as an example.

In the following, we show how to maintain the four tree structures $\mathcal{T}_i$ as θ increases from 0 to $\pi/2$.

Events. As θ increases, our trees $\mathcal{T}_i$, including the pointers, undergo changes in their structures and values storing in them. We distinguish such changes into two types of events:

- *Hull events* occur when the combinatorial structure of OH_θ changes. There are again two different types of changes: when a point is added to or removed from OH_θ or when an overlap between two opposite staircases is created or destroyed. Note that two opposite staircases st_θ and $\mathsf{st}_{\theta+\pi}$ may overlap and hence OH_θ introduce a self-crossing.
- *Pointer events* correspond to changes of pointers p_u^j. Such an event occurs exactly when the corresponding ray of p_u^j for some leaf node u of $\mathcal{T}_i$ and $j \in \{1,2,3\}$ hits a point on OH_θ, and thus its destination needs to be updated. A pointer event is associated with its occurring orientation $\phi \in [0, \pi/2)$, the corresponding leaf node u and pointer p_u^j.

From previous work on maintaining OH_θ, we can precompute all hull events.

Lemma 2 ([7, 9, 13]) *There are at most $O(n)$ hull events and all hull events can be computed in $O(n \log n)$ time using $O(n)$ space.*

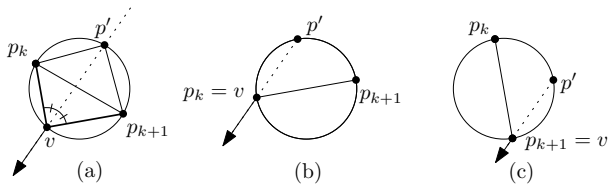


Figure 3: The line ℓ containing the ray from v in direction $\theta + \pi/4$ always goes through v regardless of θ. (a) $v \neq p_k$ and $v \neq p_{k+1}$, (b) $v = p_k$, (c) $v = p_{k+1}$.

Predicting pointer events can be done as follows. Suppose the tree structures $\mathcal{T}_i$ together with the pointers are correctly maintained until $\theta = \theta_0 \in [0, \pi/2)$. Fix a leaf node u of $\mathcal{T}_i$. For each $j \in \{1,2,3\}$, let u_j be the leaf node of $\mathcal{T}_{i+j}$ such that p_u^j points to u_j. At this moment, we can compute the possible pointer event corresponding to the next change of p_u^j after θ_0, which we will call the *candidate* event for p_u^j, denoted by $\mathrm{PECAND}_j(u, \theta_0)$. Below, we assume $i = 0$ without loss of generality, as the other cases $i = 1, 2, 3$ can be handled symmetrically.

1. For p_u^1, its corresponding ray γ hitting $\widehat{u_1}$ emanates from the vertex v of $\widehat{u}$ in direction $\theta_0 + \pi$, unless it points to null. Though the vertex v moves as θ

varies, observe that the line extending the corresponding ray γ from v always goes through one of the two endpoints of $\widehat{u}$, say $p \in P$. Hence, we can precompute when the ray γ hits one of the endpoints of $\widehat{u_1}$ and which leaf node neighboring u_1 will be the next target of p_u^1, unless there is a hull event beforehand. This constitutes the candidate event for p_u^1 after θ_0, that is, $\text{PECAND}_1(u, \theta_0)$.

2. For p_u^2, its corresponding ray γ hitting $\widehat{u_2}$ emanates from the vertex v of $\widehat{u}$ in direction $\theta_0 + 5\pi/4$, unless it points to null. Consider the two endpoints p_1 and p_2 of $\widehat{u}$ and the circle O with diameter p_1p_2. Let p' be the midpoint of the circular arc between p_1 and p_2 that avoids v. Let ℓ be the directed line extending γ. As θ increases, γ and ℓ move. In this case, we observe that ℓ rotate around a fixed point p' as well: Since the step $\widehat{u}$ makes the right angle at v, v moves along the circle O as θ varies. Now observe that the line ℓ bisects the right angle at v, since ℓ is in direction $\theta + 5\pi/4$. So, ℓ always goes through p' by the inscribed angles of circles. See Figure 3. Hence, we can compute the candidate event $\text{PECAND}_2(u, \theta_0)$ for p_u^2 after θ_0.
3. The case of p_u^3 is similar to that of p_u^1.

As described above, we can compute in $O(1)$ time the candidate pointer event $\text{PECAND}_j(u, \theta_0)$ for each pointer p_u^j at a specific moment θ_0.

Our algorithm to maintain the trees $\mathcal{T}_i$ is run by properly handling events. As a preprocessing, we compute hull events by Lemma 2 and insert them into a priority queue, called the *event queue*. We then initialize the $\mathcal{T}_i$'s for $\theta = 0$ as follows:

Initialization. First, we compute the four staircases st_0, $\mathsf{st}_{\pi/2}$, st_π, and $\mathsf{st}_{3\pi/2}$, and so OH_0. We then build $\mathcal{T}_i$ for all $i \in \{1, 2, 3, 4\}$ with the internal pointers. Observe that when traversing the leaf nodes u of $\mathcal{T}_i$ along the internal pointers, the leaf nodes in $\mathcal{T}_{i+j}$ pointed by p_u^j also change along the internal pointers of $\mathcal{T}_{i+j}$ for $j \in \{1, 2, 3\}$. Thus, all external pointers can be computed in $O(n)$ additional time. This takes $O(n \log n)$ time using $O(n)$ space in total.

Next, we compute the candidate pointer event $\text{PECAND}_j(u, 0)$ for each external pointer p_u^j, and insert it into the event queue. Since there are $O(n)$ leaf nodes in total, this can be done in additional $O(n \log n)$ time. Note that the number of (candidate) events stored in the event queue is bounded also by $O(n)$.

Handling the next event. Our algorithm then extract the next event from the event queue and handle it correctly to maintain the invariants. Let ϕ be the orientation of the next event extracted from the event queue.

If it is a pointer event $\text{PECAND}_j(u, \theta_0)$, we perform the following. Note that the event was a candidate event for pointer p_u^j at the moment $\theta = \theta_0$. This pointer event is associated with the next target leaf that p_u^j should point after ϕ. Hence, updating p_u^j can be done in $O(1)$ time with the access to the node u. Then, we compute a new candidate event $\text{PECAND}_j(u, \phi)$ for pointer p_u^j after ϕ as described above, and insert it into the event queue. This completes the procedure to handle a pointer event, spending $O(\log n)$ time.

For the total number of pointer events handled, we prove the following.

Lemma 3 *The total number of pointer events created and handled during the execution of the algorithm is bounded by $O(n^2)$.*

Therefore, all pointer-event-related operations can be done in $O(n^2 \log n)$ time.

Next, consider the case of a hull event. By Lemma 2, there are only $O(n)$ hull events. So, we can handle each hull event in a brute force way: reinitialize the structure for $\theta = \phi$ as described above. This spends $O(n \log n)$ time per hull event, and so $O(n^2 \log n)$ time in total.

Therefore, we conclude the following.

Lemma 4 *As θ increases from 0 to $\pi/2$, we can maintain the tree structures $\mathcal{T}_i$ for $0 \le i \le 3$ with the pointers in total $O(n^2 \log n)$ time using $O(n)$ space.*

Querying crosses. While the trees $\mathcal{T}_i$ are being maintained, we can test whether a given cross encloses P.

Lemma 5 *At any moment $\theta \in [0, \pi/2)$, while maintaining the $\mathcal{T}_i$'s as above, we can decide if any given cross of orientation θ encloses P in $O(\log n)$ time.*

4 Algorithm

Now we present an algorithm to compute a minimum-width minimal cross. First, we observe the following.

Lemma 6 *Let $C = \sigma_1 \cup \sigma_2$ be a minimum-width minimal cross. Exactly one of the following is true:*

- $\mathrm{w}(\sigma_1) > \mathrm{w}(\sigma_2)$ *and one of the bounding lines of σ_1 contains two points of P on the boundary of C.*
- $\mathrm{w}(\sigma_1) < \mathrm{w}(\sigma_2)$ *and one of the bounding lines of σ_2 contains two points of P on the boundary of C.*
- $\mathrm{w}(\sigma_1) = \mathrm{w}(\sigma_2)$.

Using Lemma 6, we present two different algorithms for computing a minimum-width cross, depending on whether one of the bounding lines of the cross contains two points of P on the boundary of the cross, or each of the bounding lines of the cross contains exactly one point of P on the boundary of the cross. Both of the algorithms work by increasing θ from 0 to $\pi/2$ and maintaining the trees $\mathcal{T}_i$.

4.1 When one of the bounding lines contains two points

In this section, we present an algorithm that computes a minimum-width cross when one of its bounding line contains two points of P on its boundary. Let $C = \sigma_1 \cup \sigma_2$ be the cross of orientation $\theta \in [0, \pi/2)$ such that the orientation of σ_1 is θ. We assume that the upper bounding line of σ_1 contains two points of P on the boundary of C and $\mathrm{w}(\sigma_1) \geq \mathrm{w}(\sigma_2)$, since the other cases can be handled symmetrically. By Lemma 1, both of the points on the bounding line appear on OH_θ.

Consider two points p_1 and p_2 of P such that both of the points are on OH_θ and the line ℓ going though p_1 and p_2 is of orientation θ for some $\theta \in [0, \pi/2)$. Let $C(p_1, p_2) = \sigma_1 \cup \sigma_2$ be the minimal cross with minimum width of orientation θ such that the upper bounding line of σ_1 is ℓ and $\mathrm{w}(\sigma_1) \geq \mathrm{w}(\sigma_2)$. Note that there is a point on the lower bounding line of σ_1 that appears in OH_θ by Lemma 1. We can compute $C(p_1, p_2)$ with our data structures at θ as follows. Let $q_1, \ldots, q_k$ be the points of P on OH_θ lying below ℓ, ordered in the decreasing order of y-coordinate in the coordinate system rotated by θ. For each index $1 \leq j \leq k$, we define $\sigma_1(j) = \sigma_\theta(\{p_1, q_j\})$ and $\sigma_2(j) = \sigma_{\theta+\pi/2}(P \setminus \sigma_1(j))$. Observe that as j increases, $\mathrm{w}(\sigma_1(j))$ increases while $\mathrm{w}(\sigma_2(j))$ is non-increasing. Therefore, by finding the smallest index j such that $\mathrm{w}(\sigma_1(j)) \geq \mathrm{w}(\sigma_2(j))$, we obtain $C(p_1, p_2) = \sigma_1(j) \cup \sigma_2(j)$.

For a fixed q_j, $\mathrm{w}(\sigma_1(j))$ can be computed trivially in $O(1)$ time. To compute $\mathrm{w}(\sigma_2(j))$, we must find the point q_M with the maximum x-coordinate and the point q_m with the minimum x-coordinate in $P \setminus \sigma_1(j)$, in the coordinate system rotated counterclockwise by θ. Let p_M be the point in P with the maximum x-coordinate. If $\sigma_1(j)$ does not contain p_M, then $q_M = p_M$. Otherwise, the value $x(q_M)$ is determined by the intersection $\mathsf{OH}_\theta \cap \ell$ or $\mathsf{OH}_\theta \cap \ell'$, where ℓ' is the line of orientation θ passing through q_j. The intersection $\mathsf{OH}_\theta \cap \ell$ can be found in $O(1)$ time via the step containing p_1 and the external pointer of the node storing that step. Likewise, the intersection $\mathsf{OH}_\theta \cap \ell'$ can be found in $O(1)$ time. Thus, $x(q_M)$ can be computed in $O(1)$ time, and by a similar argument, $x(q_m)$ can also be computed in $O(1)$ time. Hence, $\mathrm{w}(\sigma_2(j))$ can be computed in $O(1)$ time. We observe that the steps containing $q_1, \ldots, q_k$ are stored in the trees $\mathcal{T}_i$ for $i \in 0, 1, 2, 3$ at orientation θ, ordered in their y-coordinates. Therefore, by applying the binary search on the trees $\mathcal{T}_i$, we can compute $C(p_1, p_2)$ in $O(\log n)$ time.

As θ increases from 0 to $\pi/2$, the orientation when two points p_1 and p_2 of P on OH_θ lie on a line of orientation θ corresponds to either a hull event or a pointer event. More specifically, if p_1 and p_2 belong to the same staircase, the event is a hull event; otherwise, it is a pointer event. The total number of tree and pointer events is $O(n^2)$, and for each event, the corresponding cross $C(p_1, p_2)$ can be computed in $O(\log n)$ time. No additional data structures are required beyond the trees $\mathcal{T}_i$. Thus, by Lemma 4, we can conclude the following.

Lemma 7 *We can compute a minimum-width cross with one of the bounding line containing two points of P on the boundary of cross in $O(n^2 \log n)$ time and $O(n)$ space.*

4.2 When each of the bounding lines contains exactly one point

In this section, we present an algorithm that computes a minimum-width cross such that each of the bounding lines contains exactly one point of P on the boundary of the cross. Then it consists of the strips of the same width by Lemma 6. Let $C_\theta(p, q, r, s)$ be the cross of orientation $\theta \in [0, \pi/2)$ such that

- p and q lie on the upper and lower bounding line of the strip of orientation θ, respectively, and
- r and s lie on the left and right bounding line of the strip of orientation $\theta + \pi/2$

for $p, q, r, s \in P$. By definition, $C_\theta(p, q, r, s) = \sigma_\theta(\{p, q\}) \cup \sigma_{\theta+\pi/2}(\{r, s\})$. Since $\mathrm{w}(\sigma_\theta(\{p, q\}))$ and $\mathrm{w}(\sigma_{\theta+\pi/2}(\{r, s\}))$ are both sinusoidal functions of θ, they are equal for at most one or for all $\theta \in [0, \pi/2)$.

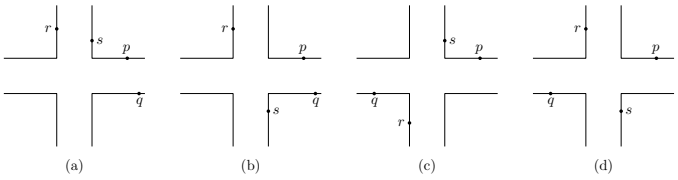


Figure 4: Four types of cross: (a) type 1, (b) type 2, (c) type 3, and (d) type 4.

Consider a cross $C = C_\theta(p, q, r, s)$ such that p, q, r, and s are the only points of P lying on the boundary of C. Then we can classify C into following four types under the symmetry group. See Figure 4.

1. $p \in h_0(C)$, $q \in h_7(C)$, $r \in h_2(C)$, and $s \in h_1(C)$.
2. $p \in h_0(C)$, $q \in h_7(C)$, $r \in h_2(C)$, and $s \in h_6(C)$.
3. $p \in h_0(C)$, $q \in h_4(C)$, $r \in h_5(C)$, and $s \in h_1(C)$.
4. $p \in h_0(C)$, $q \in h_4(C)$, $r \in h_2(C)$, and $s \in h_6(C)$.

There can be $2^4 = O(1)$ different configurations with respect to the position of p, q, r, and s on the boundary of C, but since each of them can be transformed into one of the following four types under the rotation and reflection, we present an algorithm computing a minimum-width cross belonging to one of the types above. Other case can be handled symmetrically.

Lemma 8 *There is no minimum-width cross of type 4.*

By Lemma 8, we only have to consider the crosses of type 1, 2, and 3. By Lemma 1, if a cross C of orientation θ and type $i \in \{1,2,3\}$ encloses P, then the point of P lying on $h_0(C)$ is on st_θ.

For each point $p \in P$ on st_θ and $i \in \{1,2,3\}$, we define $C^i_\theta(p)$ of orientation θ and type i such that p is the point of P on $h_0(C^i_\theta(p))$ which may be a minimum-width cross as follows.

Definition of $C^1_\theta(p)$. If p is the leftmost point of st_θ, $C^1_\theta(p)$ is not defined. Otherwise, let $s^1_\theta(p)$ be the point of P on st_θ making the step together with p on the left side of p. Let u be the leaf node of $\mathcal{T}_0$ such that $\widehat{u}$ is the step. We set $r^1_\theta(p)$ to be the point incident to the step pointed by p^1_u on the upper side. If p^1_u points to null, we set $r^1_\theta(u)$ to be the point of P at $\mathsf{st}_{\theta+\pi/2} \cap \mathsf{st}_{\theta+\pi}$. We also set $q^1_\theta(p)$ to be the point incident to the step pointed by p^3_u on the right side. If p^3_u points to null, we set $q^1_\theta(p)$ to be the point of P at $\mathsf{st}_{\theta+\pi} \cap \mathsf{st}_{\theta+3\pi/2}$. Then $C^1_\theta(p) := C_\theta(p, q^1_\theta(p), r^1_\theta(p), s^1_\theta(p))$.

Definition of $C^2_\theta(p)$. We first give following observation on a minimum-width cross of type 2.

Lemma 9 *If $C = C_\theta(p', q', r', s')$ is a minimum-width cross of orientation θ and type 2, then q' has the larger x-coordinate than p' in the coordinate system rotated by θ in counterclockwise direction.*

Note that if $C_\theta(p', q', r', s')$ is a cross of orientation θ and type 2 enclosing P, q' and s' makes a step in $\mathsf{st}_{\theta+3\pi/2}$. Moreover, Lemma 9 implies that the ray from p' in direction $\theta + 3\pi/2$ hits $\mathsf{st}_{\theta+3\pi/2}$ at the step of q' and s'.

Now we define $C^2_\theta(p)$. Let u and u' be the steps of st_θ incident to p on the left and right, respectively. Then we set $r^2_\theta(p)$ to be the point incident to the step pointed by p^1_u on the upper side, If p^1_u points to null, we set $r^2_\theta(u)$ to be the point of P at $\mathsf{st}_{\theta+\pi/2} \cap \mathsf{st}_{\theta+\pi}$. We set $q^2_\theta(p)$ and $s^2_\theta(p)$ to be the points incident to the step pointed by $\mathsf{p}^3_{u'}$ on the upper and lower side, respectively. If $\mathsf{p}^3_{u'}$ points to null, $C^2_\theta(u)$ is not defined. Otherwise, $C^2_\theta(p) := C_\theta(p, q^2_\theta(p), r^2_\theta(p), s^2_\theta(p))$.

Definition of $C^3_\theta(p)$. Assume that there is a cross $C = C_\theta(p', q', r', s')$ which is a minimum-width cross of orientation θ and type 3. Then the step of s' and p' is present at st_θ, and the step of r' and q' is present at $\mathsf{st}_{\theta+\pi}$. Let v be the vertex of the step of s' and p', and v' be the vertex of the step of r' and q'. Then v is also the vertex of $Q_0(C)$ and v' is also the vertex of $Q_2(C)$. Since C consists of strips of the same width, the line going through v and v' is of orientation $\theta + \pi/4$.

Now, we define $C^3_\theta(p)$. If p is the leftmost point of st_θ, $C^3_\theta(p)$ is not defined. Otherwise, let $s^3_\theta(p)$ be the point of P on st_θ making the step together with p on the left side of p. Let u be the leaf node of $\mathcal{T}_0$ such that $\widehat{u}$ is the step. Then we set $q^3_\theta(p)$ and $r^3_\theta(p)$ to be the points of P incident to the step pointed by p^2_u on the right and left, respectively. If p^2_u points to null, $C^3_\theta(p)$ is not defined. Otherwise, $C^3_\theta(p) := C_\theta(p, q^3_\theta(p), r^3_\theta(p), s^3_\theta(p))$.

Algorithm. As θ increases, we handle *cross events* to find a minimum-width cross. Cross events occur at θ when two strips of $C^i_\theta(p)$ for some p on st_θ and $i \in \{1,2,3\}$. The detailed algorithm is as follows.

When $\theta = 0$ and whenever a hull event occurs, we compute $q^i_\theta(p)$, $r^i_\theta(p)$, and $s^i_\theta(p)$ for all p on st_θ and $i \in \{1,2,3\}$ as an initialization. We define $\theta^i(p)$ to be the orientation in $[0, \pi/2)$ such that the widths of the cross $C^i_\theta(p)$ are equalized at $\theta = \theta^i(p)$ for $p \in P$ on st_θ and $i \in \{1,2,3\}$. If $C^i_\theta(p)$ is not defined, no such orientation exists, or the widths are the same for all orientation in $[0, \pi/2)$, $\theta^i(p)$ is not defined. We maintain all the $\theta^i(p)$ values that is larger than current θ into a *cross event queue* in the form of priority queue.

When θ reaches $\theta^i(p)$ for some p and i, we handle the cross event. In the cross event, we test whether $C^i_\theta(p)$ encloses P. This can be done in $O(\log n)$ time by Lemma 5. If it encloses P, we consider $C^i_\theta(p)$ as a candidate of a minimum-width cross. By reporting a candidate cross with the minimum width, the algorithm correctly computes a minimum-width cross.

For each hull event, computing $q^i_\theta(p)$, $r^i_\theta(p)$, and $s^i_\theta(p)$ for $i \in \{1,2,3\}$ takes $O(1)$ time for each p. Then we compute the cross event queue of $O(n)$ values accordingly, which takes $O(n \log n)$ time. Thus, it takes $O(n^2 \log n)$ time in total for hull events.

For each pointer event, let u be the leaf node in $\mathcal{T}_0$ such that its external pointer is updated in the pointer event, and let $p_1, p_2 \in P$ be the points making the step $\widehat{u}$. Then $q^i_\theta(p)$, $r^i_\theta(p)$, and $s^i_\theta(p)$ for $p \in \{p_1, p_2\}$ and $i \in \{1,2,3\}$ might be changed. Then we update $\theta^i(p)$ values accordingly, and update the cross event queue. Since we only update $O(1)$ values at once, it takes $O(\log n)$ time for each pointer event. Therefore, it takes $O(n^2 \log n)$ time in total for pointer events.

Since $O(n)$ new $\theta^i(p)$ values are updated for each hull event, and $O(1)$ new values are updated for each pointer event, there $O(n^2)$ cross events occur in total. Moreover, since we store $O(1)$ values for each $p \in P$ on st_θ in the cross event queue, it uses $O(n)$ space. Therefore, together with Lemma 4, we have the following.

Lemma 10 *We can compute a minimum-width cross with each of the bounding line containing exactly one point of P on the boundary of cross in $O(n^2 \log n)$ time and $O(n)$ space.*

By Lemma 7 and 10, we conclude the following.

Theorem 11 *For a set P of n points in $\mathbb{R}^2$, we can compute a minimum-width cross in $O(n^2 \log n)$ time and $O(n)$ space.*

References

[1] P. K. Agarwal, C. M. Procopiuc, and K. R. Varadarajan. A (1+ ε)-approximation algorithm for 2-line-center. *Computational Geometry*, 26(2):119–128, 2003.

[2] P. K. Agarwal, C. M. Procopiuc, and K. R. Varadarajan. Approximation algorithms for a k-line center. *Algorithmica*, 42:221–230, 2005.

[3] P. K. Agarwal and M. Sharir. Planar geometric location problems and maintaining the width of a planar set. In *Proceedings of the 2nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1991)*, pages 449–458. SIAM, 1991.

[4] P. K. Agarwal and M. Sharir. Efficient randomized algorithms for some geometric optimization problems. *Discrete Comput Geom*, 16:317–337, 1996.

[5] T. Ahn and S. W. Bae. Constrained two-line center problems. In *International Symposium on Algorithms and Computation*, pages 5:1–5:17, 2024.

[6] T. Ahn, C. Chung, H.-K. Ahn, S. W. Bae, O. Cheong, and S. D. Yoon. Minimum-width double-slabs and widest empty slabs in high dimensions. *Computational Geometry*, 129:102173, 2025.

[7] C. Alegría-Galicia, D. Orden, C. Seara, and J. Urrutia. On the $\mathcal{O}_\beta$-hull of a planar point set. *Computational Geometry*, 68:277–291, 2018. Special Issue in Memory of Ferran Hurtado.

[8] S. W. Bae. Minimum-width double-strip and parallelogram annulus. *Theoretical Computer Science*, 833:133–146, 2020.

[9] S. W. Bae, C. Lee, H.-K. Ahn, S. Choi, and K.-Y. Chwa. Computing minimum-area rectilinear convex hull and L-shape. *Computational Geometry*, 42(9):903–912, 2009.

[10] T. M. Chan. Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus. *International Journal of Computational Geometry and Applications*, 12(1-2):67–85, 2002.

[11] C. Chung, T. Ahn, S. W. Bae, and H.-K. Ahn. Parallel line centers with guaranteed separation. *Computational Geometry*, 129:102185, 2025.

[12] A. K. Das, S. Das, and J. Mukherjee. Approximation algorithms for orthogonal line centers. *Discrete Applied Mathematics*, 338:69–76, 2023.

[13] J. M. Díaz-Báñez, M. A. López, M. Mora, C. Seara, and I. Ventura. Fitting a two-joint orthogonal chain to a point set. *Computational Geometry*, 44(3):135–147, 2011.

[14] A. Glozman, K. Kedem, and G. Shpitalnik. On some geometric selection and optimization problems via sorted matrices. *Computational Geometry: Theory and Applications*, 11(1):17 – 28, 1998.

[15] J. W. Jaromczyk and M. Kowaluk. The two-line center problem from a polar view: A new algorithm and data structure. In *Workshop on Algorithms and Data Structures*, pages 13–25. Springer, 1995.

[16] M. J. Katz and M. Sharir. An expander-based approach to geometric optimization. *SIAM Journal on Computing*, 26(5):1384–1408, 1997.

[17] N. Megiddo and A. Tamir. On the complexity of locating linear facilities in the plane. *Operations research letters*, 1(5):194–197, 1982.

[18] G. T. Toussaint. Solving geometric problems with the rotating calipers. In *Proceedings of the 2nd Mediterranean Electrotechnical Conference (IEEE Melecon 1983)*, volume 83, page A10, 1983.