Partial Enclosure Range Searching*

Gregory Bint[1] Anil Maheswari[1] Subhas C. Nandy[2] Michiel Smid[1]

[1] School of Computer Science, Carleton University, Canada,
{gbint, anil, michiel@scs.carleton.ca}
[2] Indian Statistical Institute, Kolkata, India, nandysc@isical.ac.in

Abstract: A new type of range searching problem, called the *partial enclosure range searching problem*, is introduced in this paper. Given a set of geometric objects S and a query region Q, our goal is to identify those objects in S which intersect the query region Q by at least a fixed proportion of their original size. Two variations of this problem are studied. In the first variation the objects in S are line segments and the objective is to count the total number of members of S so that their intersection with Q is at least a given proportion of their size. Here, Q can be an axis-parallel rectangle or a slab of arbitrary orientation. In the second variation, S is a polygon and Q is an axis-parallel rectangle. The problem is to report the area of the intersection between the polygon S and a query rectangle Q.

1 Introduction

In a geometric range searching problem, a set of geometric objects S are given, such as points, lines, circles, or boxes, and the query is with respect to another well-defined geometric object Q. The objective is to identify all elements in S contained within the query region Q. Traditionally, preprocessing schemes are developed to build a data structure so that queries can be answered efficiently. Over the past four decades, several variants of range searching problems have been studied depending on the complexity of the objects in S, the query region Q, and the query requirements.

In this paper, we address a different variation of this problem, called *partial enclosure range searching (PERS)*. To the best of our knowledge, this problem is not studied previously. In this setting, the goal is to identify, for a given query region Q, all objects in S that satisfy the *partial enclosure property*. An object in S is said to satisfy the *partial enclosure property* if at least some fixed proportion of the object (with respect to its length, area, volume) intersects the query region Q.

*Research supported by NSERC.

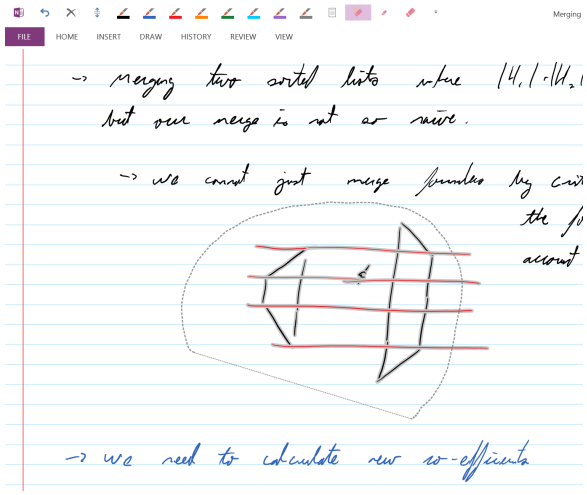


Figure 1: An example of partial enclosure range searching in Microsoft OneNote. The selected line segments are not entirely enclosed in the query region.

This problem was inspired by the use of Microsoft OneNote. Using a digital pen, OneNote can be used much like a paper notebook, allowing the user to add handwriting, diagrams, equations, etc. to a page. Figure 1 shows some handwritten notes, and a diagram which has been partially selected. Here the horizontal line segments of the diagram are not entirely enclosed by the selection tool, but they appear as part of the set of selected items. This behaviour of selecting partially enclosed objects is described in a patent [6]. Although the problems that we will examine take place in simpler settings, we will nevertheless develop an understanding of the major challenges of this problem domain, as well as some techniques for addressing them.

The paper proposes algorithms for the following variations of the partial enclosure range searching problem.

PERS - Partial enclosure range searching: Here S is given as a set of line segments, and the query object Q is an axis-parallel rectangle or a slab bounded by two parallel lines of arbitrary orientation. The objective is to count the number of objects in S that partially (or fully) lie in Q. We have considered different combinations depending on the orientation of the line segments in S and the orientation of the rectangle/slab Q.

PEAC - Partial enclosure area computation: Here S is a monotone or arbitrary polygon, and Q is an axis-parallel rectangle or slab respectively. The objective is to compute the area of the region $S \cap Q$.

Table 1 gives a broad overview of the proposed results in this paper. Here, 'AP' is used for *Axis-Parallel*, 'AO' for *Arbitrary Orientation*, and 'P' for *Polygon*.

Table 1: Summary of Contributions

Problem	Object	Query	Space	Time	Query time
PERS	AP Segment	AP Rectangle	$O(n \log^3 n)$	$O(n \log^3 n)$	$O(\log^3 n)$
PERS	AO Segment	AP Rectangle	$O(n \log^7 n)$	$O(n \log^7 n)$	$O(\sqrt{n} \log^7 n)$
PERS	AP Segment	AO Slab	$O(n \log^2 n)$	$O(n \log^3 n)$	$O(\sqrt{n} \log^3 n)$
PERS	AP Segment	2 AO Slabs	$O(n \log^3 n)$	$O(n \log^3 n)$	$O(\sqrt{n} \log^3 n)$
PEAC	Monotone P	AP Rectangle	$O(n \log n)$	$O(n \log n)$	$O(\log n)$
PEAC	Simple P	Horiz Slab	$O(n)$	$O(n)$	$O(\log n)$

The next four sections of the paper cover partial enclosure range searching queries on successively more sophisticated geometric objects. In Section 2, we state some preliminaries. Section 3 focuses on the partial enclosure range counting problem when the objects are line segments and the query region is an axis-parallel rectangle, Section 4 considers the counting problem in the same environment where the query region is an arbitrary-oriented slab. Section 5 considers the partial enclosure area computation problem in polygons. Finally, we conclude in Section 6, where we summarize our contributions and future work.

2 Preliminaries

Let P be a set of n points in $\mathbb{R}^d$ ($d \geq 2$). A range tree for P is a data structure that supports counting and reporting the members of P in an axis-parallel rectangular query range [5, Chapter 5]. It can be constructed in $O\big(n \log^{d-1} n\big)$ time using $O\big(n \log^{d-1} n\big)$ space. The counting and reporting query time complexities are $O\big(\log^{d-1} n\big)$ and $O\big(\log^{d-1} n + k\big)$ respectively, where k is the number of reported points.

The range trees are used to query ranges which can be expressed by a single query variable expression. In some cases in this paper, our ranges will take the form of half-planes comprised of two query variables, and for those, we use the following structure described in Chan[2], restated here with $d = 2$.

Theorem 1 (Corollary 7.3, part i, in Chan[2])**.** *Consider a set P of n points in the plane. We can form $O(n)$ canonical subsets of total size $O(n \log n)$ in $O(n \log n)$ time, such that the subset of all points inside any query simplex can be reported as a union of disjoint canonical subsets C_i with $\sum_i \sqrt{|C_i|} \leq O(\sqrt{n} \log n)$ in time $O(\sqrt{n} \log n)$ with high probability with respect to n.*

This structure is particularly well-suited to multi-part queries where we need to identify objects which satisfy a half-plane query, and which also satisfy some other arbitrary query condition for which we have an existing query method. To support such a query, we construct a canonical subsets structure to perform the half-plane component of the query. Then, with the objects of each canonical subset, we construct secondary query structures.

A similar structure with slightly worse query time bounds by Matousek[7] is presented in [5, Chapter 16], along with a method for using it to construct multi-part queries.

A multi-level query is executed by first querying the canonical subsets structure for all points satisfying the half-plane. With each subset identified by the first step, we then evaluate the associated secondary structure. The result is the set of all objects which satisfy both the half-plane query *and* whatever conditions the second-level query may test. If the secondary structure is itself a multi-level structure, then this procedure effectively adds one more layer, and we can repeat this process to any arbitrary number of levels. The following corollary formalizes the use of canonical subset structures to build multi-level structures and details the preprocessing and query requirements.

Corollary 1.1. *Given a set of n objects $a_1, a_2, \ldots, a_n$ which can be queried with a data structure A requiring preprocessing space $S(n)$, preprocessing time $T(n)$, and query time $Q(n)$, and where each a_i also has an associated point p_i in the plane, we can construct a multi-level data structure which can identify all objects whose associated point p_j satisfies a half-plane* and *where a_j satisfies a query on A.*

1. *If $S(n) \in O\big(n \log^f n\big)$, $T(n) \in O(n \log^g n)$, and $Q(n) \in O\big(\sqrt{n} \log^h n\big)$, where $f, g, h \in O(1), 0 \leq f \leq g$, then the resulting multi-level data structure requires $O\big(n \log^{f+1} n\big)$ preprocessing space, $O\big(n \log^{g+1} n\big)$ preprocessing time, and $O\big(\sqrt{n} \log^{h+1} n\big)$ query time with high probability.*

2. *If $S(n) \in O\big(n \log^f n\big)$, $T(n) \in O(n \log^g n)$, and $Q(n) \in O\big(\log^h n\big)$, where $f, g, h \in O(1), 0 \leq f \leq g$, then the resulting multi-level data structure requires $O\big(n \log^{f+1} n\big)$ preprocessing space, $O\big(n \log^{g+1} n\big)$ preprocessing time, and $O\big(\sqrt{n} \log^{h+1} n\big)$ query time with high probabily.*

Proof. The preprocessing space of the multi-level structure requires $O(n \log n)$ space for the canonical subsets structure itself. In addition, the associate structures need $\sum_{i=1}^{k} S(|C_i|)$ space, which is less than or equal to $\sum_{i=1}^{k} O\big(|C_i| \log^f |C_i|\big) \leq O\Big(\log^f n \cdot \sum_{i=1}^{k} |C_i|\Big)$ (since $|C_i| \leq n$ for all i) $\leq O\big(\log^f n \cdot n \log n\big)$ (by Theorem 1). Thus, the total space complexity is $O\big(n \log^{f+1} n\big)$. Preprocessing time is calculated in the same manner, resulting in the time complexity of $O\big(n \log^{g+1} n\big)$.

Querying requires $O(\sqrt{n} \log n)$ time with high probability to find the k' disjoint canonical subsets representing the elements found by the top-level canonical subsets query, plus the time required to query the associated data structures.

If $Q(n) \in O\big(\sqrt{n} \log^h n\big)$ then the total time to query the appropriate associated structures is $\sum_{i=1}^{k'} Q(|C_i|) \leq \sum_{i=1}^{k'} O\Big(\sqrt{|C_i|} \log^h |C_i|\Big) \leq O\Big(\log^h n \cdot \sum_{i=1}^{k'} \sqrt{|C_i|}\Big) \leq O\big(\log^h n \cdot \sqrt{n} \log n\big)$ (by Theorem 1).

Otherwise, if $Q(n) \in O\big(\log^h n\big)$ then the total time to query the appropriate associated structures is $\sum_{i=1}^{k'} Q(|C_i|) \leq \sum_{i=1}^{k'} O\big(\log^h |C_i|\big) \leq O\Big(\sum_{i=1}^{k'} \log^h n\Big) \leq O\Big(\log^h n \cdot \sum_{i=1}^{k'} 1\Big)$

$\leq O\big(\log^h n \cdot \sqrt{n}\log n\big)$ (by Theorem 1).
Thus, the total query time is $O\big(\sqrt{n}\log^{h+1} n\big)$ with high probability. □

We use this theorem and corollary as "black boxes" in Sections 3 and 4.

3 PERS problem for Axis-Parallel Rectangles

We present two variations of the PERS problem in this section. In the first variation (Subsection 3.1), the line segments in S will be axis-parallel, whereas in the next variation (Subsection 3.2), we allow the segments in S to have arbitrary orientation. In both the variations, the query rectangle Q is axis-parallel. We first define the concept of partial enclosure of segments as follows:

Definition 1. *A segment $s \in S$ is said to satisfy the* partial enclosure property *with respect to a query object Q if and only if $|s \cap Q| \geq \rho \cdot |s|$, where $|x|$ denotes the length of the segment x and $0 < \rho \leq 1$.*

3.1 Axis-Parallel Segments

In this section, we consider the following problem.

Problem 1. *Given a set S of n axis-parallel line segments in the plane, and a fixed parameter ρ ($0 < \rho \leq 1$), we want to identify those segments which are sufficiently enclosed by an axis-parallel query rectangle Q so as to satisfy the partial enclosure property.*

We use the following notation. Each segment $s_i = (a_i, b_i, \ell_i) \in S$, $1 \leq i \leq n$ is defined by its left or bottom endpoints (a_i, b_i) depending on whether s_i is horizontal or vertical, and its length ℓ_i. The query rectangle Q is given by its bottom-left corner (α, β) and its top-right corner (γ, δ). We say that $s_i \in_\rho Q$ if and only if s_i satisfies the partial enclosure property w.r.t. Q, otherwise $s_i \notin_\rho Q$.

For horizontal segments, we only need to consider the segments $s = (a, b, \ell)$ satisfying $\beta \leq b \leq \delta$. Figure 2 illustrates several cases regarding how such a segment may interact with Q. Cases (1), (2), and (3) demonstrate the cases where s is entirely to the left, entirely within, or entirely to the right of Q, respectively. Case (4) considers the situation where s crosses only the left boundary of Q (i.e., $\alpha \leq a + \ell \leq \gamma$). Depending on the partial enclosure parameter ρ, we further subdivide case (4) into subcases ($4a$) if $s \in_\rho Q$, and ($4b$) if $s \notin_\rho Q$. Cases ($5a$) and ($5b$) are similar to cases ($4a$) and ($4b$), but with respect to γ. Specifically, s falls into case ($5a$) or ($5b$) when $\alpha \leq a \leq \gamma$ and $a + \ell > \gamma$. In case (6), s crosses both the left and right boundaries of Q, with neither of its endpoints inside Q; the subcases are ($6a$) if $s \in_\rho Q$ and ($6b$) if $s \notin_\rho Q$. Our goal is to identify all segments belonging to cases (2), ($4a$), ($5a$), and ($6a$), and none of the segments belonging to any other case.

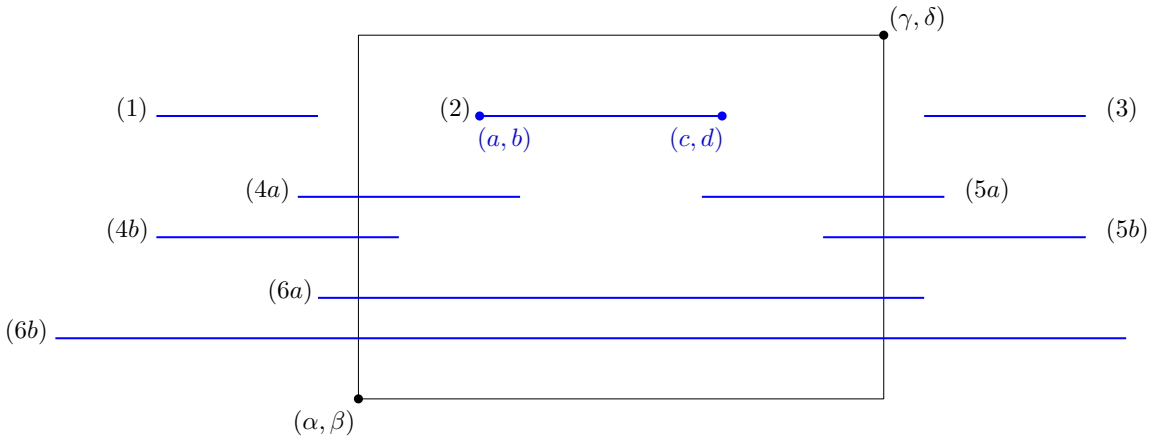


Figure 2: An axis-parallel query on axis-parallel segments. Different cases of horizontal segments interacting with the query region are shown.

Let $w = \gamma - \alpha$ be the width of Q. We need to consider only the segments in $S_1 = \{s_i \in S \mid \beta \leq b_i \leq \delta \ \& \ \ell_i \leq \frac{w}{\rho}\}$, discarding all segments in case $(6b)$, among others. We partition the members in S_1 according to the location of their left endpoint with respect to α. Specifically, let $S_L = \{s_i \in S_1 \mid a_i < \alpha\}$ and let $S_R = \{s_i \in S_1 \mid a_i \geq \alpha\}$. Now, we test an appropriate partial enclosure expression to determine whether s_i should be counted. For segments in S_L, we want to ensure that "not too much of s_i is outside of Q", i.e., $S'_L = \{s_i \in S_L \mid \alpha - a_i < (1 - \rho) \cdot l_i\}$. For segments in S_R, we want to ensure that "enough of s_i is inside Q", i.e., $S'_R = \{s_i \in S_R \mid \gamma - a_i \geq \rho l_i\}$.

Observation 1. *The subset of segments satisfying the partial enclosure property is* $S_\rho = S'_L \cup S'_R$.

The members in S'_L can be identified by mapping each segment $s_i \in S$ to a point $\hat{s}_i = (b_i, \rho \cdot \ell_i, a_i, a_i + (1 - \rho) \cdot \ell_i)$ in $\mathbb{R}^4$, and then observing those points lying inside the four dimensional query box $\hat{Q} = [\beta, \delta] \times (0, w] \times (-\infty, \alpha] \times (\alpha, \infty)$. Similarly, the members in S'_R can be identified by mapping each segment $s_i \in S$ to a point $\hat{\hat{s}}_i = (b_i, \rho \cdot \ell_i, a_i, a_i + \rho \cdot \ell_i)$ in $\mathbb{R}^4$, and then observing those points lying inside the four dimensional query box $\hat{\hat{Q}} = [\beta, \delta] \times (0, w] \times [\alpha, \infty) \times (-\infty, \gamma]$.

We can answer these queries by constructing two 4D range trees [5] with two sets of points $\{\hat{s}_i | s_i \in S\}$ and $\{\hat{\hat{s}}_i | s_i \in S\}$ respectively, and executing the counting query with the corresponding 4D query rectangle. The preprocessing time and space required for constructing these two range trees are both $O\big(n \log^3 n\big)$, and the query can be answered in $O\big(\log^3 n\big)$ time. Finally, we report the sum of two results as the answer of the query. Note that we can use the same first three levels for both the range trees since the first three components of both the types of query points (for $a < \alpha$ and $a > \alpha$) are same. In the third level, we create *two* associated structures, one for each of the partial enclosure expressions, and query the one as needed.

For vertical segments, the method of querying is exactly similar to that for horizontal segments, only we need to consider the height of Q instead of its width, and we consider symmetric coordinates of each segment while mapping them to points in 4D. The following theorem summarizes the solution.

Theorem 2. *Given a set of n axis-parallel line segments, we can identify a set of disjoint subsets containing all segments which satisfy the partial enclosure property for an axis-parallel query rectangle in $O(\log^3 n)$ time, with a data structure requiring $O(n \log^3 n)$ preprocessing time and space.*

3.2 Arbitrarily-Oriented Segments

In this subsection, we allow each segment $s_i \in S$ to have any arbitrary orientation. They may intersect among themselves. Here, the partial enclosure problem is stated as follows:

Problem 2. *Given a set of n arbitrarily-oriented line segments in the plane, and a fixed parameter ρ such that $1/2 < \rho \leq 1$, we want to identify those segments which are sufficiently enclosed by an axis-parallel query rectangle Q so as to satisfy the partial enclosure property. A segment s satisfies this property if and only if $|s \cap Q| \geq \rho \cdot |s|$.*

We use $s_i = [(a_i, b_i), (c_i, d_i)]$ to denote a segment, where (a_i, b_i) and (c_i, d_i) are two endpoints of s_i satisfying $a_i \leq c_i$, and if $a_i = c_i$ then $b_i < d_i$. The problem statement is as follows.

3.2.1 Decomposing the Problem

We have three principal cases to consider: those segments which have both endpoints inside Q, those with only one endpoint inside Q, and those with both endpoints outside Q which still intersect Q.

We first construct a 4D range tree $\mathcal{T}$ with the points (a_i, b_i, c_i, d_i) for each $s_i \in S$, and execute a 4D range query with query box $[\alpha, \gamma] \times [\beta, \delta] \times [\alpha, \gamma] \times [\beta, \delta]$ corresponding to $Q = [\alpha, \beta] \times [\gamma, \delta]$ to identify all the segments satisfying the different cases. The first case, where both the end-points are inside Q, will all satisfy the partial enclosure property.

For the second case, we apply the same method, but on a *virtual* segment as follows. For each $s_i = [(a_i, b_i), (c_i, d_i)] \in S$, create two virtual segments s_i' and s_i'', where $s_i' \subseteq s_i$, s_i' has an endpoint (a_i, b_i), with $|s_i'| = \rho \cdot |s_i|$, and $s_i'' \subseteq s_i$, s_i'' has an endpoint (c_i, d_i), with $|s_i''| = \rho \cdot |s_i|$. Thus, if $s_i' \in Q$ and/or $s_i'' \in Q$, then $s_i \in_\rho Q$. Counting both cases may double-count some segment(s) which have both endpoints in Q, so we subtract the count accordingly. As in the previous case, we can solve this case using a 4D range query.

The last case, where both the endpoints of a segment are outside Q, is the most challenging case to handle. We begin by partitioning the space outside Q into 8 regions by extending lines through its horizontal and vertical boundaries. These regions are labelled

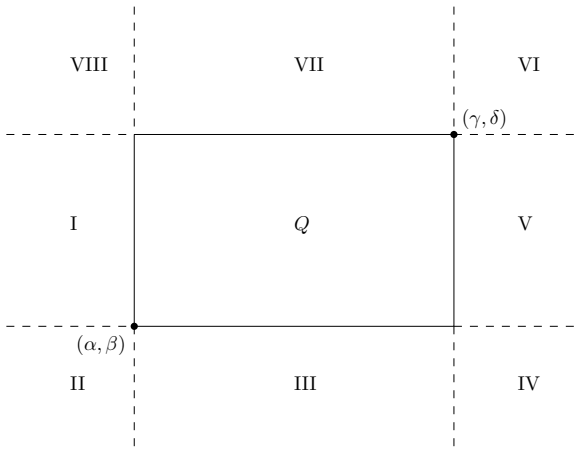


Figure 3: The 8 regions surrounding Q.

as $I, II, \ldots, VIII$ in anticlockwise order starting from the left-middle region (see Figure 3). Any segment which passes through Q but has neither endpoint in Q will have its endpoints in two distinct regions. Not every pair of regions is legal[1].

A segment which passes through Q may involve in any one of the following four classes depending on the pair of regions containing its two endpoints:

(i) In two parallely opposite cells: $(I, V), (III, VII)$.

(ii) In two non-corner cells adjacent to a corner of Q: $(I, III), (III, V), (V, VII), (VII, I)$.

(iii) In a corner cell and a non-corner cell: (II, V), (II, VII), (IV, I), (IV, VII), (VI, I), (VI, III), $(VIII, III)$, $(VIII, V)$.

(iv) In two corner cells: $(II, VI), (IV, VIII)$.

We will query for each class separately and combine our results at the end. In the first phase, we identify each class of problem by performing an appropriate rectangular range searching in the data structure $\mathcal{T}$. This also indicates which partial enclosure expressions we need to test in the second phase of the query. We develop expressions for each case below. In all the cases, we require the following additional definitions. Let $o_i = (e_i, f_i)$ be the mid-point of s_i. We use $d(u, v)$ to denote the Euclidean distance between two points $u, v \in \mathbb{R}^2$. Note that, for our choice of ρ, if $o_i \notin Q$, then s_i does not satisfy the partial enclosure property. Thus, we assume that $o \in Q$. we use p' and q' to represent the points of intersection of the segment s with Q closest to p and q respectively.

Class (i): This case deals with the segments that cross the parallel sides of Q. We present the details of the partial enclosure property for the subcase where $p = (a, b) \in I$ and $q = (c, d) \in V$; the solution for the subcase (III, VII) is similar.

[1]For example, a segment with its two endpoints in regions I and II respectively, cannot intersect Q.

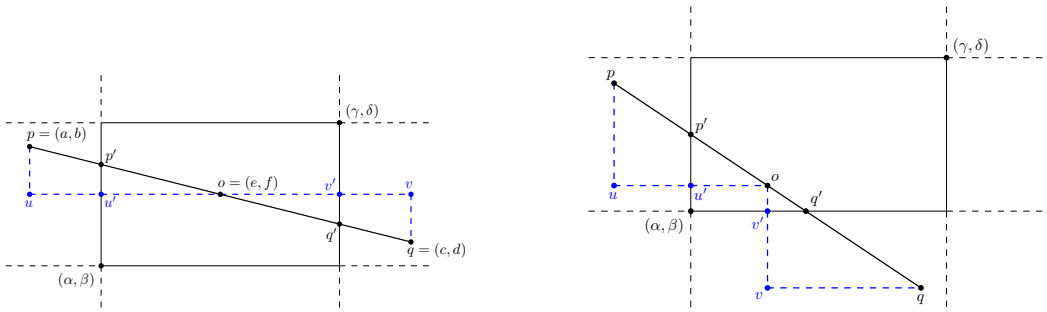


Figure 4: Demonstration of (a) Case A and (b) Case B

Assume for now that $o \in Q$. Let p' and q' be the points of intersection of the segment s with Q closest to p and q respectively. Let us draw a right angle triangle Δpuo whose edge $[u, o]$ intersects the boundary of Q at $u' = (\alpha, f)$. Similarly, the edge $[v, o]$ of Δqvo intersects the boundary of Q at $v' = (\gamma, f)$ (see Figure 4(a)). The segment s satisfies the partial enclosure property if $\frac{d(p,p')+d(q,q')}{d(p,q)} \leq 1 - \rho$. Let $L = \frac{d(p,q)}{2}$. By similarity of triangles Δpuo and $\Delta p'u'o$, we have $\frac{d(p,p')}{d(p,o)} = \frac{d(p,p')}{L} = \frac{d(u,u')}{d(u,o)} = \frac{2d(u,u')}{2d(u,o)} = \frac{2(\alpha-a)}{c-a}$. Similarly, from the similarity of triangles Δqvo and $\Delta q'v'o$, we have $\frac{d(q,q')}{d(q,o)} = \frac{2(c-\gamma)}{c-a}$.

Thus, $\frac{d(p,p')+d(q,q')}{d(p,q)} = \frac{d(p,p')+d(q,q')}{2L} = \frac{1}{2}\left(\frac{d(p,p')}{L} + \frac{d(q,q')}{L}\right) = \frac{\alpha-a}{c-a} + \frac{c-\gamma}{c-a} = 1 + \frac{\alpha-\gamma}{c-a}$.

Therefore, the inequality $\frac{d(p,p')+d(q,q')}{d(p,q)} \leq 1 - \rho$ can instead be evaluated as $1 + \frac{\alpha-\gamma}{c-a} \leq 1 - \rho$, which is based on two query variables α and γ. We can further simplify the expression to $\rho(c - a) \leq \gamma - \alpha$, where the value of $\gamma - \alpha$ is a single query variable expression calculated at query time. This inequality can therefore be checked by augmenting the search tree $\mathcal{T}$ with another level with respect to the variable $h = \rho(c - a)$ corresponding to the segments in S, and querying with $h \leq \gamma - \alpha$ at the relevant nodes in that level of $\mathcal{T}$. Note that, if $o \notin Q$, then s does not satisfy the partial enclosure property, and this test will also fail. So, no test is required to check whether $q \in Q$.

Class (ii) This case is concerned with the segments which cross the mutually perpendicular sides of Q. We present the details of the partial enclosure property for the case where $p \in I$ and $q \in III$; the solutions for the other subcases are similar.

Again let us assume that $o \in Q$ [2]. As earlier, let p' and q' be the points of intersection of s with the boundary of Q closest to p and q respectively. As in Class (i), here also we draw the right-angle triangles Δpuo and $\Delta p'u'o$. In Δpuo, let $u' = (\alpha, f)$ be the point of intersection of the line segment $[u, o]$ with the boundary of Q, and $\Delta p'u'o$ is similar to Δpuo. Similarly, in Δqvo, the point $v' = (e, \beta)$ is the intersection point of $[v, o]$ with the boundary of Q, where $\Delta q'v'o$ is similar to Δqvo. See Figure 4(b). We need to identify those

[2]If $o \notin Q$, s will not satisfy the partial enclosure property, and it can be shown that the test derived for this case will fail.

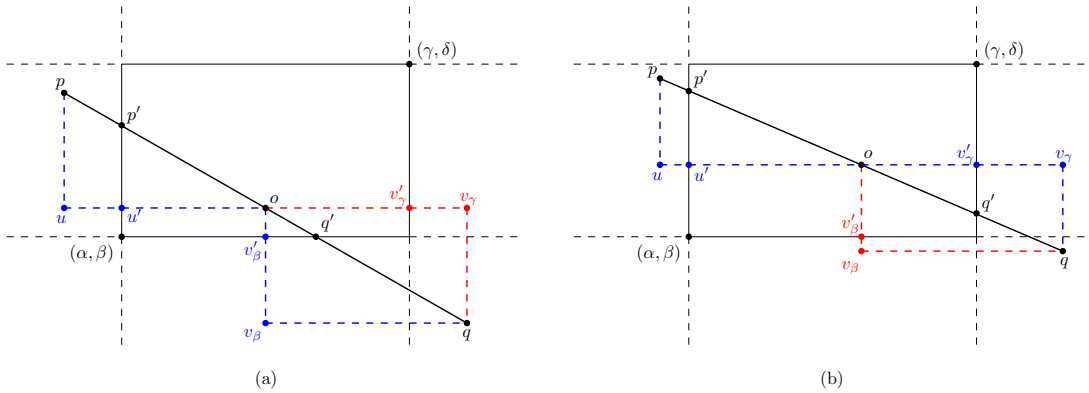


Figure 5: Example of segments in subcase (I, IV) of class (iii): the blue follows the proper handling, while the red shows the incorrect handling.

segments satisfying $\frac{d(p,p')+d(q,q')}{d(p,q)} \leq 1-\rho$.

By similarity of triangles, we have $\frac{d(p,p')}{d(p,o)} = \frac{d(p,p')}{L} = \frac{d(u,u')}{d(u,o)} = \frac{2(\alpha-a)}{c-a}$, and similarly, $\frac{d(q,q')}{d(q,o)} = \frac{2(\beta-d)}{b-d}$. Thus, $\frac{d(p,p')+d(q,q')}{d(p,q)} = \frac{d(p,p')+d(q,q')}{2L} = \frac{1}{2}\left(\frac{d(p,p')}{L} + \frac{d(q,q')}{L}\right) = \frac{\alpha-a}{c-a} + \frac{\beta-d}{b-d}$.

Therefore, the inequality $\frac{d(p,p')+d(q,q')}{d(p,q)} \leq 1-\rho$ can be tested by checking the equivalent inequality $\frac{\alpha-a}{c-a} + \frac{\beta-d}{b-d} \leq 1-\rho$, defined on the two query variables α and β. On simplification, it becomes $\alpha + \beta \cdot \frac{c-a}{b-d} \leq \left((1-\rho) + \frac{d}{b-d}\right) \cdot (c-a) + a$.

Thus, we can map each segment s satisfying Class (ii) to a point in the dual plane given by

$$(z, w) = \left(\frac{c-a}{b-d}, \left((1-\rho) + \frac{d}{b-d}\right) \cdot (c-a) + a\right)$$

The segments matching the partial enclosure expression correspond to the points satisfying the half-plane $y \geq \beta x + \alpha$ in the dual plane. In order to perform this test, we need to add a half-plane query data structure (ham-sandwich cut tree) [2] with the points (z, w) corresponding to the points in S at the leaf level of $\mathcal{T}$ (independent of the arrangement made in Class (i)), and querying with the halfplane obtained from the query interval Q as the desired node in the leaf level of $\mathcal{T}$.

Class (iii) Here, each segment has one endpoint in a corner region of Q, and the other endpoint in a non-corner region of Q as shown in Figure 5. We present the details of the partial enclosure property for the subcase where $p \in I$ and $q \in IV$; the solutions for other subcases of this case are similar.

Consider the situation where $o \in Q$. We need to consider two subcases: (a) s crosses the lines $x = \alpha$ and $y = \beta$ and (b) s crosses the lines $x = \alpha$ and $x = \gamma$. See Figures 5(a) and 5(b) for examples. Subcase (a) is very close to the example presented for class (ii). In that case, the only use of the fact that the endpoints of s were located in regions I and III was to imply that s crossed the lines $x = \alpha$ and $y = \beta$. As such, this subcase can use the same expression for testing the partial enclosing property for the segment s. Likewise, Subcase

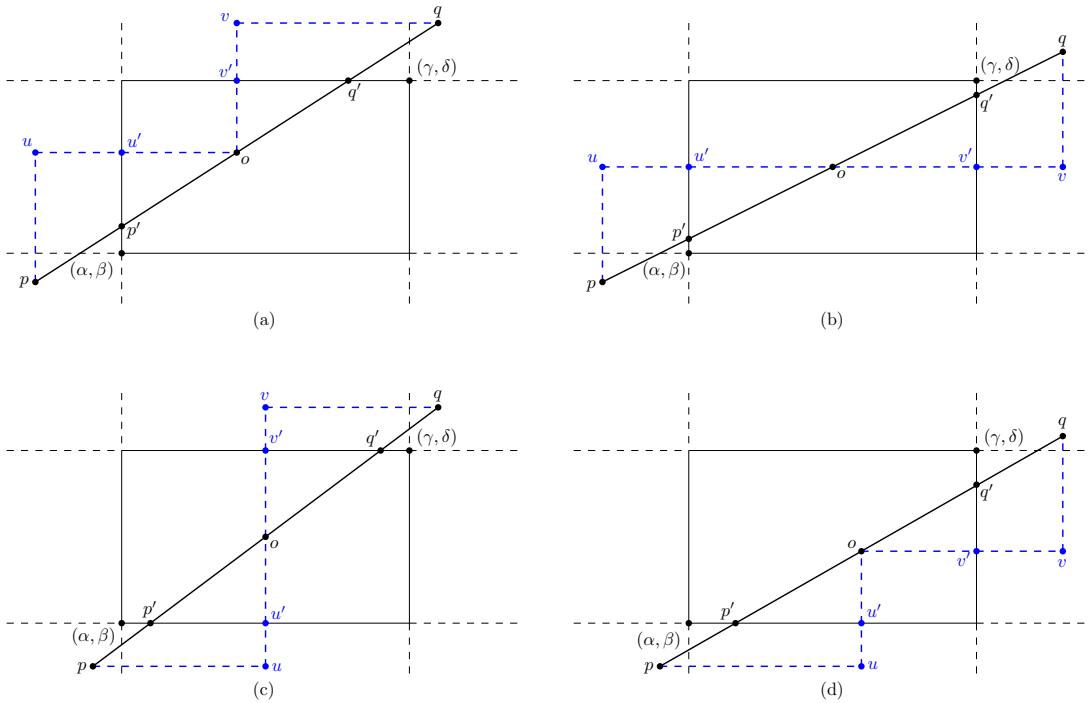


Figure 6: Example of segments of subcase case (II, VI) of class (iv): in each figure, the blue lines show the proper handling.

(b) is similar to the example presented for class (i) and can use exactly that expression for testing s.

Our initial query for identifying the regions of p and q does not allow us to differentiate between the subcases, so we will check both subcases simultaneously. From class (i) and class (ii), we have the following expressions: $\frac{\alpha-a}{c-a} + \frac{c-\gamma}{c-a} \leq 1-\rho$ and $\frac{\alpha-a}{c-a} + \frac{\beta-d}{b-d} \leq 1-\rho$ respectively. If s belongs to subcase (a), then $\frac{c-\gamma}{c-a} \leq \frac{\beta-d}{b-d}$ since γ is farther right than the point where s exits from Q. Likewise, in subcase (b), $\frac{\beta-d}{b-d} \leq \frac{c-\gamma}{c-a}$. Therefore, in either subcase, the result of the correct expression is larger than the incorrect one, allowing us to correctly reject segments by blindly checking both conditions. This also holds when $o \notin Q$, since the expression for at least one subcase would exclude the segment.

Class (iv) This case is concerned with segments whose endpoints appear in diagonally opposite corner regions of Q. We present the details of the partial enclosure property for the case where $p \in II$ and $q \in VI$; the solution for subcase $(VI, VIII)$ is similar.

Assume for now that $o \in Q$. As in class (iii), here also we need to consider several subcases depending on which sides of Q are intersected by s, namely (a) s crosses the lines $x = \alpha$ and $y = \delta$, (b) s crosses the lines $x = \alpha$ and $x = \gamma$, (c) s crosses the lines $y = \beta$ and $y = \delta$, and (d) s crosses the lines $y = \beta$ and $x = \gamma$ (see Figure 6 for examples of each subcase).

Our query in the first phase for identifying the regions of p and q does not allow us to

differentiate between subcases. However, it is sufficient to check all subcases simultaneously. The expression for subcase (b) comes directly from class (i). Using similar techniques as we did for classes (i) and (ii), we develop the following set of partial enclosure expressions:

$$(a) \quad \frac{\alpha - a}{c - a} + \frac{d - \delta}{d - b} \leq 1 - \rho \qquad (b) \quad \frac{\alpha - a}{c - a} + \frac{c - \gamma}{c - a} \leq 1 - \rho$$

$$(c) \quad \frac{\beta - b}{d - b} + \frac{d - \delta}{d - b} \leq 1 - \rho \qquad (d) \quad \frac{\beta - b}{d - b} + \frac{c - \gamma}{c - a} \leq 1 - \rho$$

Subcases (b) and (c) can be simplified to orthogonal range queries, while in subcases (a) and (d), each segment needs to be mapped to a point in an appropriate plane and, as in class (ii), the halfplane counting query needs to be performed for answering the query problem.

To illustrate that checking all four conditions simultaneously yields correct results, consider what happens if s belongs to subcase (a), where s crosses α and δ. In these circumstances, u' is above β, and we have that $\frac{\alpha-a}{c-a} \geq \frac{\beta-b}{d-b}$. Likewise, v' is left of γ, so $\frac{d-\delta}{d-b} \geq \frac{c-\gamma}{c-a}$. Therefore, the expression for (a) dominates the other expressions (i.e., (a) $\geq$ (b), (a) $\geq$ (c), and (a) $\geq$ (d)), allowing us to identify qualifying segments for subcase (a) by blindly checking all conditions. This property is true for segments belonging to subcases (b), (c), and (d) as well. Furthermore, this test also holds when $o \notin Q$, since the expression for at least one subcase would exclude the segment.

3.3 Construction and Analysis

Our method takes a very case-by-case approach to solving the problem, and executes in two phases. The first phase must classify segments belonging to one of the interesting classes among (i) to (iv), and then its appropriate subclass by searching the data structure $\mathcal{T}$. In the second phase, the partial enclosure property is tested by querying in the respective augmented data structures of $\mathcal{T}$ in its last level.

Broadly, our solution to this problem uses a multi-level range tree [5] for the classification phase, and a combination of range trees and canonical subsets structures [2] to check the partial enclosure expressions.

Identifying the segments which satisfy each case is a matter of testing a set of several conditions, all of which must be true. The order that we test the conditions in makes no difference to the correctness of the algorithm, but can have an impact on its space requirements. We now summarize the total time and space complexity for solving this problem.

Class (i) - subcase $(p, q) \in (I, IV)$**:** We need to find those segments whose end-points are in the appropriate regions and satisfy the partial enclosure expression. For each $s_i \in S$,

we construct a 5D point $v_i = (a_i, b_i, c_i, d_i, \rho(c_i - a_i))$. For the query, we construct a 5D box $Q' = (-\infty, \alpha) \times [\beta, \delta] \times (\gamma, \infty) \times [\beta, \delta] \times (-\infty, \gamma - \alpha]$ and search for the points in the box. The following lemma summarizes the complexity results.

Lemma 1. *Given a set S of n line segments, we can identify the subset of S belonging to class (i)* and *satisfying the partial enclosure property in $O\big(\log^4 n\big)$ time using a data structure requiring $O\big(n \log^4 n\big)$ preprocessing time and space.*

Class (ii) - subcase $(p, q) \in (I, III)$: As with all cases, part of the problem involves first identifying those segments with endpoints in the appropriate regions. In this case, however, the partial enclosure expression is evaluated using a half-plane query.

The classification portion is performed just as we have seen above. We map each segment to the 4D point $v_i = (a_i, b_i, c_i, d_i)$. For each $s_i \in S$, we define $h_i = \left(\frac{c-a}{b-d}, \left((1-\rho) + \frac{d}{b-d}\right) \cdot (c-a) + a\right)$. We need to construct a data structure which can answer queries on pairs (v_i, h_i). We can query the classification component using a range tree according to the following lemma.

Lemma 2. *Given a set of n line segments, we can identify a set of disjoint subsets containing all segments belonging to class (ii) in $O\big(\log^3 n\big)$ time using a data structure requiring $O\big(n \log^3 n\big)$ preprocessing time and space.*

By Theorem 1, we can query the half-plane component of our query objects using a canonical subsets data structure, giving the following lemma.

Lemma 3. *Given a set of n line segments, we can identify a set of disjoint subsets containing all segments satisfying a half-plane representation of a partial enclosure expression in $O(\sqrt{n} \log n)$ time, using a data structure requiring $O(n \log n)$ preprocessing time and space.*

Since the order that we check our conditions in does not affect correctness, we will check the half-plane condition first, then the endpoint classification. By Corollary 1.1, we can accomplish this by associating a classification structure with each subset of the half-plane structure. The resulting structure is summarized by the following lemma.

Lemma 4. *Given a set of n line segments, we can identify a set of disjoint subsets containing all segments belonging to Class (ii)* and *which satisfy their partial enclosure property in $O\big(\sqrt{n} \log^4 n\big)$ time using a data structure requiring $O\big(n \log^4 n\big)$ preprocessing time and space.*

Class (iii) - subcase $(p, q) \in (I, IV)$: This case can use the same orthogonal structure that we developed in Lemma 1 and the same half-plane expression as in Lemma 3. One component of the query box needs to be updated to account for region IV, giving the following:

$$(-\infty, \alpha) \times [\beta, \delta] \times (\gamma, \infty) \times (-\infty, \beta) \times (-\infty, \gamma - \alpha]$$

By Corollary 1.1, we can combine these two data structures to create a new structure which can answer this case as summarized by the following lemma.

Lemma 5. *Given a set of n line segments, we can identify a set of disjoint subsets containing all segments belonging to class (iii)* and *which satisfy their partial enclosure property in $O\big(\sqrt{n}\log^5 n\big)$ time using a data structure requiring $O\big(n\log^5 n\big)$ preprocessing time and space.*

Class (iv) - subcase $(p, q) \in (II, VI)$: This case has the largest number of partial enclosure expressions which need checking, in addition to the usual endpoint classification step. As a result, it will be the largest data structure to build.

The basic steps are just as in the last three cases. To cover endpoint classification and the two orthogonal partial enclosure expressions, we will use a data structure and query box similar to Lemma 1, but extended to 6D to cover the extra partial enclosure expression. We then apply Lemma 3 and Corollary 1.1 twice to handle the two half-plane partial enclosure expressions and associate the orthogonal range tree. The entire structure for this case is summarized by the following lemma.

Lemma 6. *Given a set of n line segments, we can identify a set of disjoint subsets containing all segments belonging to Class (iv)* and *which satisfy their partial enclosure property in $O\big(\sqrt{n}\log^7 n\big)$ time using a data structure requiring $O\big(n\log^7 n\big)$ preprocessing time and space.*

Combining the Steps. Querying the entire environment requires us to create the structures for handling the cases where one or both endpoints of a line segment lie entirely inside a query Q, as well as the structures for each of the cases when both endpoints lie outside of Q. Overall, this process is dominated by the structure required for the class (iv) segments. The following theorem summarizes the overall solution.

Theorem 3. *Given a set of n arbitrarily-oriented line segments, we can identify a set of disjoint subsets containing all segments which satisfy the partial enclosure property for an axis-parallel query rectangle in $O\big(\sqrt{n}\log^7 n\big)$ time, using a data structure requiring $O\big(n\log^7 n\big)$ preprocessing time and space.*

4 PERS problem for Arbitrarily-Oriented Slabs

In this section, we show how we can answer partial enclosure range searching queries where the elements of S are horizontal line segments, and the query region Q is a slab bounded by two parallel lines of arbitrarily orientation. Next, we extend our solution for a query region which is the intersection of two slabs.

4.1 Querying with One Slab

In this section, we address the following problem.

Problem 3. *We are given a set S of n horizontal line segments in the plane, and a fixed parameter ρ such that $0 < \rho \leq 1$. The objective is to identify those segments which are sufficiently enclosed inside (satisfy partial enclosure property with respect to) an arbitrarily oriented slab Q.*

As in Section 3.1, here also we use a triple $s_i = (a_i, b_i, \ell_i)$ to represent an object in S. The query slab Q is given by three inputs - (α, β, w), where the left and right bounding lines of Q are defined by $L_1 : y = \alpha x + \beta$ and $L_2 : y = \alpha x + \beta - \alpha w$ respectively. Thus, w is the horizontal width of Q.

4.1.1 Identifying the Segments

Identifying whether a segment $s_i \in_\rho Q$ (that is, whether s_i satisfies the partial enclosure property w.r.t. Q), requires three broad steps:

1. Restrict segments to those which are "not too long" to fit sufficiently inside Q.
2. Classify all segments by whether their left endpoints are left or right of L_1.
3. For each class of segments, test an appropriate partial enclosure expression.

We will use a multi-level canonical sets data structure [2] for answering these queries. We now describe the different steps of the query in more detail. Subsection 4.1.5 describes the construction and analysis of the overall data structure.

4.1.2 Restrict Length

The first step of the query is to perform a length test, as it simplifies future steps. With the query parameter w given, only segments with length $\ell \leq \frac{w}{\rho}$ can satisfy the partial enclosure property. Thus, with a 1-dimensional orthogonal range to query we can extract a subset $S_1 = \{s \in S \mid \rho\ell \leq w\}$.

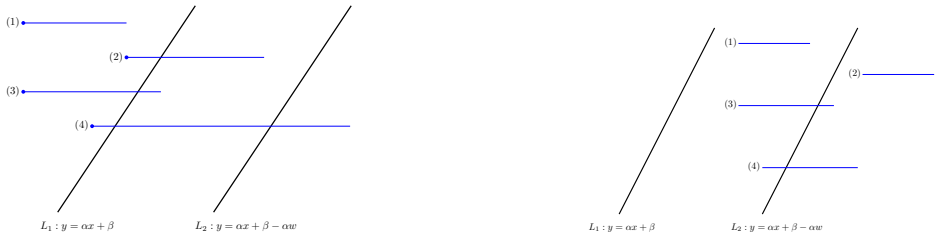
4.1.3 Classify Endpoints

The left endpoints of the members of S can be in any one of the following three regions: (1) left of L_1, (2) between L_1 and L_2, and (3) right of L_2, however, only those segments belonging to cases (1) and (2) are interesting to us. We will see that partitioning segments as left or right of L_1 is sufficient, as we can discriminate between cases (2) and (3) while testing partial enclosure expressions in the next step.

Identifying segments whose left endpoints appear to the desired side of L_1 can be accomplished using a half-plane query on the left endpoints of the members in S. Thus, we have $S_L = \{s \in S_1 \mid p \text{ is left of } L_1\}$, and let $S_R = \{s \in S_1 \mid p \text{ is right of } L_1\}$.

4.1.4 Check the Partial Enclosure Property

For each of S_L and S_R, the final step is to identify those segments which satisfy the partial enclosure property.



(a) (b)

Figure 7: (a) Segments whose left endpoints are left of L_1, (b) Segments whose left endpoints are between L_1 and L_2.

Set S_L can be classified into four subsets as follows (see Figure 7(a)):

(1) Segments which are entirely left of L_1 (and which are not counted),

(2) Segments which intersect L_1, and are sufficiently enclosed by Q,

(3) Segments which intersect L_1, but not sufficiently enclosed by Q (and which are not counted),

(4) Segments which intersect both L_1 and L_2.

Given a segment $s \in S_L$, with left endpoint $p = (a, b)$, let $\overline{s} : y = b$ be the line through s, and let (a', b) be the intersection point of $\overline{s}$ and L_1, where $a' = \frac{b-\beta}{\alpha}$.

Observation 2. *$s \in_\rho Q$ if and only if $a' - a < (1 - \rho)\ell$.*

Proof. We look at each of the above cases to show that this single expression is enough to identify all segments correctly. First, the test directly identifies segments belonging to cases (2) or (3) since $a' - a$ is precisely the amount of s outside of Q. This test rejects segments in case (1) since $a' - a > \ell > (1 - \rho)\ell$ and cannot satisfy partial enclosure property for any allowed value of ρ.

Finally, this test is also sufficient for case (4) owing to the earlier length restriction step. If a segment is not too far left of L_1, then either it crosses only L_1, and case (2) holds, or it crosses L_1 and L_2. In the latter case we know that $|s| < \frac{w}{\rho}$, where w is the width of the

query slab. Since any segment in case (4) has the property $|s \cap Q| = w$, this implies that $s \in_\rho Q$. □

Thus, among the segments in S_L, we can count those satisfying the partial enclosure property by executing a halfplane range counting query: $a' - a < (1-\rho)\ell \equiv a + (1-\rho)\ell > \frac{1}{\alpha}b - \frac{\beta}{\alpha}$. We map each segment s to a point with coordinates $(b, a + (1-\rho)\ell)$. The segments in S_L satisfying the partial enclosure expression then correspond to the points satisfying the halfplane $y > \frac{1}{\alpha}x - \frac{\beta}{\alpha}$.

Similar to S_L, the segments in S_R can be classified into four subsets, as shown in Figure 7(b).

(1) Segments which are between L_1 and L_2. Here, $s \in_\rho Q$.

(2) Segments whhich are entirely to the right of L_2. Here $s \cap Q = \emptyset$, and hence $s \notin_\rho Q$.

(3) Segments which intersect L_2, and $s \in_\rho Q$.

(4) Segments which intersect L_2, but $s \notin_\rho Q$.

Let $\overline{s}$ be the horizontal line through s, and (a'', b) be the intersection point of $\overline{s}$ with L_2 where $a'' = \frac{b-\beta}{\alpha} + w$. The following observation is easy to show.

Observation 3. *$s \in_\rho Q$ if and only if $a'' - a \geq \rho\ell$.*

As in the case of S_L, here also the counting of elements in S_R satisfying $a'' - a \geq \rho\ell$, or equivalantly $\rho\ell + a \leq \frac{1}{\alpha}b - \frac{\beta}{\alpha} + w$, can be done using a half-plane range query. Here we map each segment $s = (a, b, \ell) \in S_R$ to a point $(b, \rho\ell + a)$, and the query is performed with the halfplane $y \leq \frac{1}{\alpha}x - \frac{\beta}{\alpha} + w$.

4.1.5 Construction and Analysis

We will use the multi-level canonical sets data structure described in Section 2 to perform parts of this query. Each of the three steps given in subsection 4.1.1 will correspond to one nested level of the final data structure. It is easiest to describe the structure inside-out, so we begin with the innermost structure.

The innermost structure answers the length restriction step of the overall query. This is easily answered using a 1-dimensional range tree (AVL tree) [5], keyed on the segment lengths. Thus, we have

Lemma 7. *Given a set of n horizontal line segments, we can identify a set of disjoint subsets containing all segments with length at most $\frac{w}{\rho}$ in $O(\log n)$ time, using a data structure of size $O(n)$, which can be built in $O(n \log n)$ preprocessing time.*

To identify segments satisfying the partial enclosure property, we use the half-plane expressions as mentioned above. There are two expressions we need to test, one for S_L and the other for S_R. By Theorem 1, we can answer this type of half-plane query directly using a canonical subsets data structure, yielding the following lemma applicable to both cases.

Lemma 8. *Given a set of n horizontal line segments, we can identify a set of disjoint subsets containing all segments which are not "too much" to one side of a query line in $O(\sqrt{n}\log n)$ time, using a data structure of size $O(n\log n)$, which can be built in $O(n\log n)$ preprocessing time.*

With each subset created for this structure, we will associate the structure required for Lemma 7. Applying Corollary 1.1 gives us the following result.

Lemma 9. *Given a set of n horizontal line segments, we can identify a set of disjoint subsets containing all segments which are not "too much" to one side of a query line* and *which do not exceed a maximum length in $O\big(\sqrt{n}\log^2 n\big)$ time, using a data structure of size $O(n\log n)$, which can be built in $O\big(n\log^2 n\big)$ preprocessing time.*

We need to classify the left-endpoints of the segments as left or right of L_1. By Theorem 1, each of these queries can be answered by a canonical subsets data structure. With each subset created by this structure, we will associate the structure from Lemma 9. Applying Corollary 1.1 gives us the following result.

Lemma 10. *Given a set of n horizontal line segments, we can identify a set of disjoint subsets containing all segments whose left endpoints are to one side of a line, which are not "too much" to one side of a query line,* and *which do not exceed a maximum length in $O\big(\sqrt{n}\log^3 n\big)$ time, using a data structure of size $O\big(n\log^2 n\big)$, which can be built in $O\big(n\log^3 n\big)$ preprocessing time.*

Finally, to fully answer any query Q, we need to preprocess two such data structures, choosing our expressions appropriately for the left and right cases. During query time, we will query both structures and combine their results. The following theorem summarizes the overall solution.

Theorem 4. *Given a set of n horizontal line segments, we can identify a set of disjoint subsets containing all segments which satisfy the partial enclosure property for an arbitrarily-oriented query slab in $O\big(\sqrt{n}\log^3 n\big)$ time, using a data structure of size $O\big(n\log^2 n\big)$, requiring $O\big(n\log^3 n\big)$ preprocessing time.*

4.2 Querying with Two Slabs

In this subsection, we consider a generalization of the slab query, where the objects in S are same as in Section 4.1, and the query input are a pair of slabs; we are interested in those segments which satisfy the partial enclosure property with respect to their intersection. Formally, the problem is stated as follows:

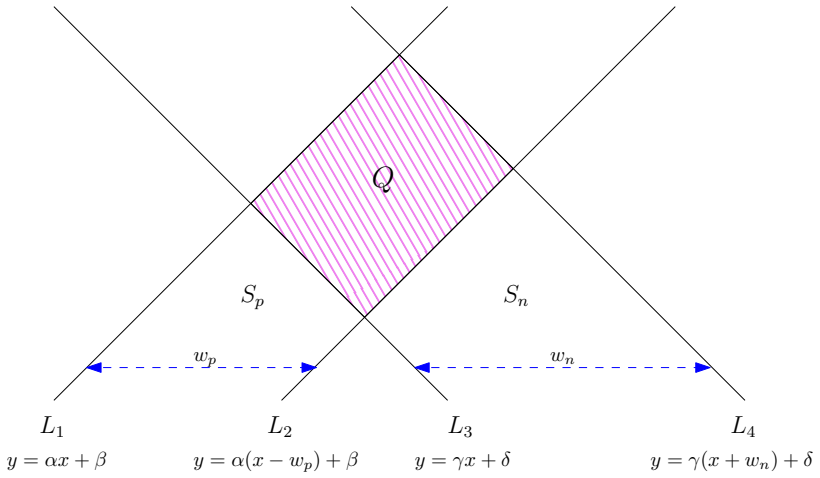


Figure 8: A query parallelogram Q formed by the inputs α, β, w_p, γ, δ, and w_n.

Problem 4. *Given a set S of n horizontal line segments in the plane and a fixed parameter ρ such that $0 < \rho \leq 1$, we want to identify those segments $s \in S$ that satisfy the partial enclosure property ($s \in_\rho Q$) with respect to a parallelogram Q, or in other words, $|s \cap Q| \geq \rho \cdot |s|$.*

If the two slabs are orthogonal, then the query is with respect to a rectangle of arbitrary orientation. The overall approach is very similar to subsection 4.1, and the data structure we will build to answer these queries is again based on the canonical sets structure outlined in Section 2.

Here the query parallelogram Q is defined by the intersection of two slabs, S_p and S_n, which have positive and negative slopes, respectively. See Figure 8.

Specifically, a query is given as a 6-tuple $(\alpha, \beta, w_p, \gamma, \delta, w_n)$, where $\alpha > 0$, $w_p > 0$, $\gamma < 0$, and $w_n > 0$. With these inputs, we define:

S_p: a slab (α, β, w_p) with positive slope α, whose left edge is defined by the line $L_1 : y = \alpha x + \beta$, and the width is w_p. Thus, its right edge is defined by $L_2 : y = \alpha(x - w_p) + \beta$.

S_n: a slab (γ, δ, w_n) with negative slope γ, whose left edge is defined by the line $L_3 : y = \gamma x + \delta$, and the width is w_n. Thus, its right edge is defined by $L_4 : y = \gamma(x + w_n) + \delta$.

Q: The parallelogram $S_p \cap S_n$.

4.2.1 Identifying the Segments

Just as in the single slab problem, identifying the segments $s_i \in_\rho Q$ is accomplished by classifying its endpoints by how they interact with the boundaries of S_p and S_n forming Q, and then testing an appropriate partial enclosure expression.

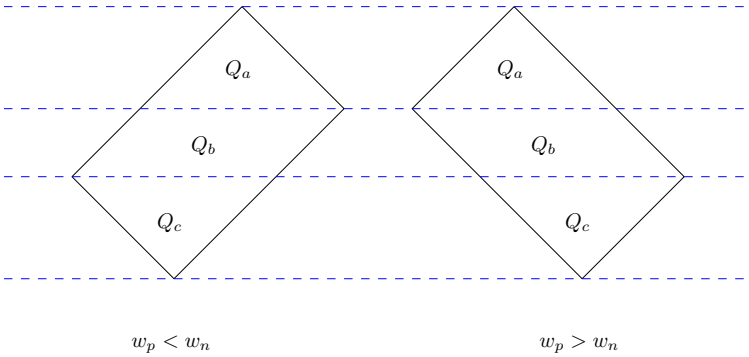


Figure 9: Decomposition of the query region Q. The orientation of Q depends on the relative widths of the slabs which define it.

The query Q is decomposed into three regions, Q_a, Q_b, and Q_c, by extending horizontal lines through each vertex of Q. Q_a and Q_c are triangular regions, while Q_b is a parallelogram. Q_a is always defined by L_1 on the left and L_4 on the right. Likewise, Q_c, is always defined by L_3 on the left and L_2 on the right. The definition of Q_b depends on the overall orientation of Q, which depends on the relationship between w_p and w_n. Specifically, Q_b is defined by L_1 and L_2 when $w_p < w_n$ and defined by L_3 and L_4 when $w_p > w_n$. When $w_p = w_n$, Q_b disappears.

4.2.2 Endpoint Classification:

Once we know the definitions of each region, we can proceed with endpoint classification. The goal of this step is to identify which partial enclosure expression is appropriate to test with each line segment s by considering the appropriate borders of Q it interacts with. Each region has its left, center, and right classification zone, Z_L, Z_C, and Z_R, where the center zone is the query region itself. Figure 10 gives an illustration; Z_L, Z_C, and Z_R are coloured blue, green, and red, respectively.

For Q_a, Z_L is aligned with the base of Q_a, and extends up L_1. Z_R is also aligned with the base of Q_a, and extends up L_4. Both of these zones are open away from Q. The zones for Q_c are symmetric, being aligned with the top of the region, and with Z_L and Z_R extending down L_3 and L_2, respectively.

For Q_b, the classification zones we use depend on the orientation of the region. When $w_p < w_n$, Z_L is aligned with the base of Q_b and extends up L_1, while Z_R is aligned with the top of Q_b and extends down L_2. Conversely, when $w_p > w_n$, Z_L is aligned with the top of Q_b and extends down L_3, while Z_R is aligned with the bottom of Q_b and extends up L_4. The former case is what is illustrated in Figure 10. Z_C is Q_b itself, which we decompose into two triangular subzones so that all of our zones are triangular in nature. During the query, any time that we consider Z_C for Q_b, we will query both subzones and consider their

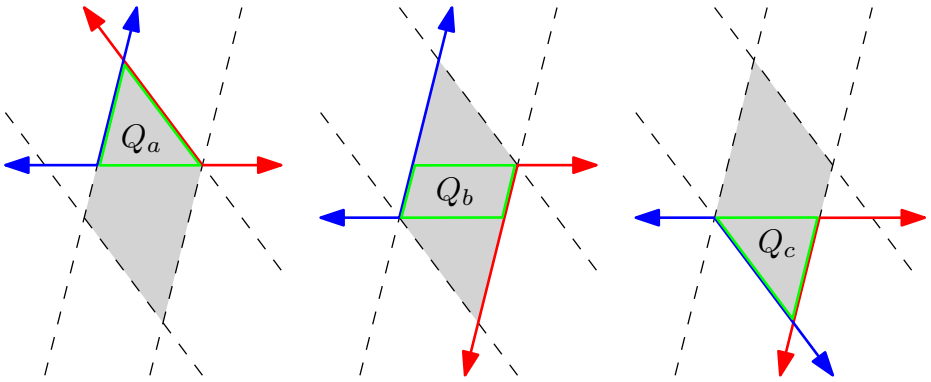


Figure 10: Classification zones for each region of Q. Each region has a left, center, and right zone, coloured blue, green, and red, respectively, where we may find segments which need further testing.

union.

Any segment interacting with a particular query region must have its endpoints in one of 6 combinations of classification zones: (Z_L, Z_L), (Z_L, Z_C), (Z_L, Z_R), (Z_C, Z_C), (Z_C, Z_R), or (Z_R, Z_R). For a segment $s \in S$ with endpoints p and q:

- If $p, q \in (Z_C, Z_C)$, then s is entirely inside Q and $s \in_\rho Q$.
- If $p, q \in (Z_L, Z_L)$, or $p, q \in (Z_R, Z_R)$ then s is entirely outside Q and $s \notin_\rho Q$.
- Otherwise, s crosses one or both boundaries of the query region and we need to test the appropriate partial enclosure expression.

4.2.3 Partial Enclosure Property:

We begin by examining Q_a in detail. Let $s \in S$ be a horizontal line segment and let $\overline{s}$ be the line through s, defined by the equation $y = b$. Let $(a', b) = \overline{s} \cap L_1$ and $(a'', b) = \overline{s} \cap L_4$, be the intersection points of $\overline{s}$ with L_1 and L_4, respectively, then $a' = \frac{b-\beta}{\alpha}$ and $a'' = \frac{b-\delta}{\gamma} - w_n$. We have three combinations of classification zones that need further testing.

For the (Z_L, Z_C) case, we know that s only crosses L_1. We check that not too much of s is outside of Q_a, giving the partial enclosure expression: $a' - a < (1-\rho)\ell$, or in other words, $b\frac{1}{\alpha} - \frac{\beta}{\alpha} < a + (1-\rho)\ell$.

We can query for segments matching this expression by performing a half-plane query on an appropriate dual-space. For this expression in particular, we map each segment s to a dual-point with coordinates $(b, a + (1-\rho)l)$, and query with the half-plane $y > \frac{1}{\alpha}x - \frac{\beta}{\alpha}$.

For the (Z_C, Z_R) case, we know that s only crosses L_4. We check that enough of s is inside Q_a, which gives the partial enclosure expression: $a'' - a \geq \rho\ell$, or in other words,

$b \cdot \frac{1}{\gamma} - \frac{\delta}{\gamma} - w_n \geq a + \rho\ell$. We can test this expression by mapping each segment s to a dual-point with coordinates $(b, a + \rho l)$ and then querying with the half-plane $y \leq \frac{1}{\gamma}x - \frac{\delta}{\gamma} - w_n$. Finally, for the (Z_L, Z_R) case, we know that both endpoints of s are outside of Q_a, so we only need to measure the width of $s \cap Q_a$. Specifically, we require that:

$$\begin{aligned} a'' - a' &\geq \rho l \\ \frac{b-\delta}{\gamma} - w_n - \frac{b-\beta}{\alpha} &\geq \rho l \\ \frac{b}{\gamma} - \frac{\delta}{\gamma} - w_n - \frac{b}{\alpha} + \frac{\beta}{\alpha} &\geq \rho l \\ b \cdot \left(\frac{1}{\gamma} - \frac{1}{\alpha}\right) + \left(\frac{\beta}{\alpha} - \frac{\delta}{\gamma} - w_n\right) &\geq \rho l \end{aligned}$$

We can test this expression by mapping each segment s to a dual-point with coordinates $(b, \rho l)$ and then querying with the following half-plane. $y \leq \left(\frac{1}{\gamma} - \frac{1}{\alpha}\right) \cdot x + \left(\frac{\beta}{\alpha} - \frac{\delta}{\gamma} - w_n\right)$. Classification into the left and right zones is somewhat "rough", as the zone continues above Q_a itself. This is not a problem in the (Z_L, Z_C) and (Z_C, Z_R) cases since one endpoint of s is classified directly in the closed zone Z_C and the segments are horizontal. However, it can happen that a segment classified into (Z_L, Z_R) is entirely above Q_a. In this case, since we are measuring $a'' - a'$, and $a'' < a'$ above the apex of Q_a, the expression will be negative and the segment will be rejected.

The partial enclosure expressions for Q_b and Q_c are developed using exactly the same reasoning as for Q_a, differing only by which of the lines L_1, L_2, L_3, and L_4 we use to define a' and a''.

4.2.4 Construction and Analysis

We will use a multi-level query structure for this problem, just as we did for the single slab query. Each of the steps given in Section 4.2.1 will correspond to one nested level of the data structure. It is easiest to describe the structure inside-out, so we begin with the innermost structure.

To identify segments satisfying a partial enclosure expression, we use the half-plane dual-spaces we developed in subsection 4.2.1. By Theorem 1, we can answer such a half-plane query directly using a canonical subsets data structure, yielding the following lemma.

Lemma 11. *Given a set of n horizontal line segments, we can identify a set of disjoint subsets containing all segments which are not "too much" to one side of a query line in $O(\sqrt{n}\log n)$ time, using a data structure of size $O(n \log n)$, which can be built in $O(n \log n)$ time.*

We need to be able to classify the right endpoints of our segments into any of the classification zones of each query region. Each of these regions are triangular in nature and

can be queried with a canonical subsets data structure. With each subset created by this structure, we will associate the structure from Lemma 11. Applying Corollary 1.1 gives us the following result.

Lemma 12. *Given a set of n horizontal line segments, we can identify a set of disjoint subsets containing all segments which have their right endpoint in a query triangle, and which are not "too much" to one side of a query line, in $O\big(\sqrt{n}\log^2 n\big)$ time, using a data structure of size $O\big(n\log^2 n\big)$, which can be built in $O\big(n\log^2 n\big)$ time.*

Next, we need to be able to classify the left endpoints of our segments into any of the classification zones as well, which will use another canonical subsets data structure. With each subset of this structure, we continue to build up our multi-level structure by associating the structure from Lemma 12. By applying Corollary 1.1 again, we have the following.

Lemma 13. *Given a set of n horizontal line segments, we can identify a set of disjoint subsets containing all segments which have their left and right endpoints in given query triangles, and which are not "too much" to one side of a query line, in $O\big(\sqrt{n}\log^3 n\big)$ time, using a data structure of size $O\big(n\log^3 n\big)$, which can be built in $O\big(n\log^3 n\big)$ time.*

Finally, to fully answer a query Q, we only need one instance of the structures for classifying the left and right endpoints, since these structures can support all of the different triangular classification queries we need to perform. We will need to preprocess several versions of the innermost level which determines the partial enclosure expression, however. During query time, we check (Z_C, Z_C), (Z_L, Z_C), (Z_C, Z_R) and (Z_L, Z_R) for every query region. For any of the latter three zone combinations which are non-empty, we check the innermost structure corresponding to the appropriate partial enclosure expression. The following theorem summarizes this process.

Theorem 5. *Given a set of n horizontal line segments, we can identify a set of disjoint subsets containing all segments which satisfy the partial enclosure property with respect to the intersection of two query slabs in $O\big(\sqrt{n}\log^3 n\big)$ time, using a data structure of size $O\big(n\log^3 n\big)$, requiring $O\big(n\log^3 n\big)$ preprocessing time.*

4.3 Remarks on Arbitrarily-Oriented Segments

Our general approach can be extended to identify arbitrarily-oriented line segments which satisfy the partial enclosure property for one or two slabs, however, the resulting structure is very "case heavy" and involves a large number of query variables.

Arbitrarily-oriented segments can cross through the borders of Q in many more ways than horizontal ones. This prevents us from partitioning the environment into a simple set of classification zones, and increases the number of cases to consider.

The partial enclosure expressions for each case will also involve higher degree polynomials. For example, a segment s which crosses L_1 and L_2 has intersection points $p' = (a', b')$ with

L_1 and $q' = (a'', b'')$ with L_2. To consider how much of the segment is inside the slab, we need to calculate the length of the segment from p' to q', as follows.

$$\begin{aligned}\|p'q'\|^2 &= \left(\sqrt{(a''-a')^2 + (b''-b')^2}\right)^2 \\ &= (a''-a')^2 + (b''-b')^2 \\ &= (a'')^2 - 2a'a'' + (a')^2 + (b'')^2 - 2b'b'' + (b')^2\end{aligned}$$

Each term in the above expression represents several query variables when we consider the actual values for a', b', a'', and b''. Let $\overline{s}$ be the line through s, and assume that it is defined as $y = mx + t$, then $a' = \frac{\beta - t}{m - \alpha}$, $b' = m\frac{\beta - t}{m - \alpha} + t$, $a'' = \frac{\beta - t - \alpha w}{m - \alpha}$ and $b'' = m\frac{\beta - t - \alpha w}{m - \alpha} + t$. Multiplying and squaring these expressions to calculate $\|p'q'\|^2$ results in a high degree polynomial with respect to the number of independent query variable expressions. The situation worsens when we consider intersections between L_1 or L_2 with L_3 or L_4, as we have to consider γ and δ as well. Such an expression *can* be mapped to a high-degree half-space and then be queried with a canonical subset structure, but the space and query times suffer.

5 PEAC Problem on Monotone Polygons

In this section, we describe the partial enclosure range searching problem on monotone polygons. Formally, the problem is stated as follows:

Problem 5. *We are given a monotone polygon P consisting of n vertices and a fixed parameter ρ such that $0 < \rho \leq 1$. We want to determine whether at least $\rho \cdot \text{area}(P)$ is enclosed by an axis-parallel query rectangle Q, where area(P) is the area of the polygon P.*

The main contribution of this section is a method to calculate the area of a monotone polygon appearing within an axis-parallel query rectangle. Once the enclosed area is calculated, determining if the partial enclosure property is satisfied is straight-forward.

Throughout this section, we will use the following definitions. The input is a polygon P embedded in the plane which, without loss of generalization, is monotone with respect to the x-axis (i.e., from left to right). P is defined by its vertices $v_1, v_2, \ldots, v_n$, where v_n and v_1 share an edge to close the polygon. A query Q is given by its lower-left and upper-right corners (α, β) and (γ, δ), respectively. See Figure 11 for an example.

5.1 A First Problem

We will first describe an algorithm which solves a simpler problem, and then extend that solution into one for our main problem.

Problem 6. *Given a monotone polygon P, and a query in the form of a horizontal slab S, what is the total area of P enclosed by S? i.e., what is* area$(S \cap P)$*?*

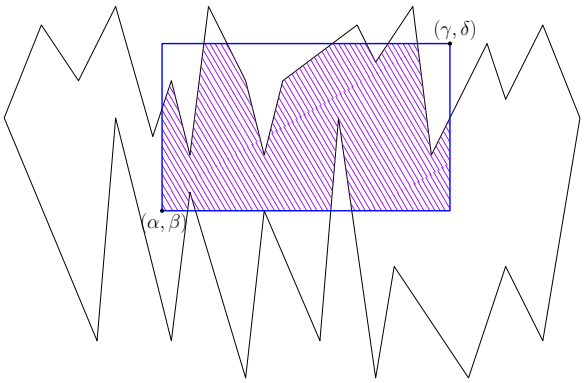


Figure 11: A monotone polygon P with query Q. The area of P enclosed by Q, $Q \cap P$, is highlighted.

Our overall approach for this problem is to define a function (actually, a collection of functions) which returns the area below a given horizontal query line.

5.1.1 Decomposing P

In order to create an area formula for the entire polygon, we decompose P into a linear number of regions and calculate area formulas for each. These regional formulas are then combined into an overarching *multi-region formula*.

We begin by sorting all of the vertices by their x-coordinates. If two vertices share the same x-coordinate (no more than two can do so, since P is monotone), we can skip the duplicate without any other change to the algorithm. However, for ease of discussion, we will assume that every vertex has a distinct x-coordinate. For the remainder of this section, we relabel the x-coordinates by their sorted order so that x_1 is the x-coordinate of the leftmost vertex of P, x_2 the next leftmost, etc.

Let $\overline{x}$ be the vertical line through any x-coordinate, and let P_U and P_L be the upper and lower chains of P, respectively. Consider the sequence of lines $\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}$ through the vertices of P. For every line $\overline{x_i}$, let $a_i = \overline{x_i} \cap P_U$ and $a'_i = \overline{x_i} \cap P_L$ be the intersections of $\overline{x_i}$ with the upper and lower chains of P, respectively.

We will calculate all of these intersections as we walk from left to right over both P_U and P_L simultaneously. We also keep track of e_i and e'_i, the edges of P_U and P_L, respectively, which are intersected by $\overline{x_i}$, and upon which reside the points a_i and a'_i. Where $\overline{x_i}$ intersects a vertex of P_U or P_L, we store the edge to the right of that vertex as the corresponding value of e_i or e'_i. This walk allows us to generate a sequence of regions $R_1, R_2, \ldots, R_{n-1}$ which have the following, equivalent definitions.

- For each $1 \leq i \leq n-1$, let X_i be the vertical slab between $\overline{x_i}$ and $\overline{x_{i+1}}$, then $R_i = X_i \cap P$.

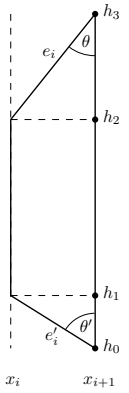


Figure 12: A trapezoidal region of P for which we can derive an area function dependant on h.

- R_i is the area of P bounded by the vertical segments $\overline{x_i} \cap P$ and $\overline{x_{i+1}} \cap P$ to the left and right, and some portion of the edges e_i and e'_i to the top and bottom. $\overline{x_{i+1}}$, this excess is irrelevant to the definition of R_i).
- R_i is the polygon formed by the cycle on $a_i, a_{i+1}, a'_{i+1}, a'_i$.

Thus, each region is a (possibly degenerate) trapezoid.

5.1.2 Area of a Region

For each region $R_i \in \{R_1, R_2, \ldots R_{n-1}\}$, we create a function $F(R_i, h)$ which will return the area of R_i below a horizontal query line having a y-component of h. Figure 12 shows an example region. From bottom to top, we can identify at most 4 "critical heights", where the nature of how the area of a region grows with respect to h changes. For a general region R, these 4 critical heights are:

- h_0, where R begins.
- h_1, where R stops growing as a triangle and begins growing as a rectangle.
- h_2, where R stops growing as a rectangle and begins growing as the base trapezoid of a triangle.
- h_3, where R ends.

These 4 critical heights give rise to a piecewise function representing the area, defined as

follows.

$$F(R,h) = \begin{cases} 0 & \text{if } h < h_0 \\ A(h')^2 & \text{if } h_0 \leq h < h_1 \text{ for } h' = h - h_0 \\ Bh' + C & \text{if } h_1 \leq h < h_2 \text{ for } h' = h - h_1 \\ A'(h')^2 + C' & \text{if } h_2 \leq h < h_3 \text{ for } h' = h_3 - h \\ C'' & \text{if } h_3 \leq h \end{cases}$$

In essence, the area function is a quadratic one, although, for different values of h, some or all of the coefficients are 0. The details of each part of this function, and the definitions of the constants A, A', B, C, C' and C'' are as follows.

1. When $h < h_0$, the area of R below h is 0, since no part of R exists below that measure.

2. When $h_0 \leq h < h_1$, the area of R below h grows as a triangle. Let $h' = h - h_0$. The angle θ' between e_i' and the vertical can be calculated as

   $$\tan(\theta') = \frac{x_{i+1} - x_i}{h_1 - h_0}$$

   Let t' be this ratio, then at h', the base of the triangle has width $h' \cdot t'$. The area of the triangle is therefore $\frac{1}{2}(h')^2 t'$ and we set the constant $A = \frac{1}{2}t'$.

3. When $h_1 \leq h < h_2$, the area of R includes all of the area between h_0 and h_1, which we store as the constant C. The remaining area of R up to h grows as a rectangle. Let $h' = h - h_1$, then the rectangular area is $(x_{i+1} - x_i) \cdot h'$, and we define the constant $B = (x_{i+1} - x_i)$. Thus, the total area of R_i up to h' is $Bh' + C$.

4. When $h_2 \leq h < h_3$, the area of R includes all of the area between h_0 and h_2, which we store as the constant C_a'. The remaining area of R up to h grows as the base trapezoid of a triangle. Let $h' = h_3 - h$ (note that this definition is somewhat reversed from the other cases). We can calculate this area by taking the total area of the triangle, and subtracting the area of the triangle from h_3 down to h.

   The overall triangle between h_2 and h_3 has area

   $$a = \frac{1}{2} \cdot (h_3 - h_2) \cdot (x_{i+1} - x_i)$$

   The angle θ between e_i and the vertical can be calculated as

   $$\tan(\theta) = \frac{x_{i+1} - x_i}{h_3 - h_2}$$

   Let t be this ratio, then the smaller triangle from h_3 down to h can be calculated as in case 2, giving $a' = \frac{1}{2}(h')^2 t$. The overall area from h_2 up to h is then

   $$a - a' = \frac{1}{2} \cdot (h_2 - h_3) \cdot (x_{i+1} - x_i) - \frac{1}{2}(h')^2 t$$

This expression is of the form $C'_b + A'(h')^2$. Thus we set the constants $A' = -\frac{1}{2}t$, $C'_b = \frac{1}{2} \cdot (h_2 - h_3) \cdot (x_{i+1} - x_i)$, and $C' = C'_a + C'_b$ to complete the formula for this part of $F(R, h)$.

5. When $h_3 \leq h$, the total area of R should be reported, which we store in C''.

5.1.3 Creating Multi-Region Formulas

After completing the previous step, we have a set of regions, and their area formulas, R_i and $F(R_i, h)$ respectively for $1 \leq i \leq n-1$. To complete the preprocessing for the horizontal slab query, we will process our regional area formulas into a list of multi-region formulas which can report the area of the entire polygon under a query line.

We begin by collecting tuples of all of the critical heights from our regional formulas. For every region R_i, we collect (y, R) for $y \in \{h_0, h_1, h_2, h_3\}$ and $R = R_i$. Let $\mathcal{Y}$ be the list of all such tuples across all regions, sorted by y-values from lowest to highest.

If we suppose for a moment that each y-value is distinct, then for each one there is exactly one region R which is transitioning from one phase of growth to another. That is, looking at the piecewise function $F(R, h)$ which determines the area of R below a query line, a new piece of that function is taking over. To create the multi-region formula for a new critical height, we copy the formula for the predecessor height (the lowest y-value starts with a formula set to 0). We update the multi-region formula by subtracting the coefficients of R for its previous phase of growth, so that they no longer influence the formula, and then add the coefficients for the new phase.

Since neighbouring regions share vertices, the y-values will not be distinct. Instead, several y-values will be collapsed into a single critical height in the list of multi-region formulas, with all appropriate constants from contributing regions added or subtracted as necessary. Algorithm 1 gives the details of this process more formally.

The output of this algorithm is the list H of multi-region formulas. Since $\mathcal{Y}$ was sorted by y-value, H will be as well.

5.1.4 Querying Multi-Region Formulas

We are now ready to query horizontal slabs against P. Let $F(P, h)$ be the multi-region area function for P, which is essentially a piecewise function given by Algorithm 1 in the form of a sorted list H. Given two horizontal lines $\overline{\beta}$ and $\overline{\delta}$ defining our query slab, which have y-components β and δ, respectively with $\beta \leq \delta$, our query is answered as follows:

1. Our query parameter β will not generally correspond to a critical height in H. Perform a binary search on H to find the formula stored with the *predecessor* of β. Denote this formula as F_β. Evaluating $F_\beta(P, \beta)$ tells us the area of P below β.

Algorithm 1: BuildMultiRegionFormula

Input: List of regions $\mathcal{R} = R_1, R_2, \ldots, R_{n-1}$

Initialize a list $\mathcal{Y}$
foreach R *in* $\mathcal{R}$ **do**
  $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(y, R) \mid y \in \{R.h_0, R.h_1, R.h_2, R.h_3\}\}$
`sort(`$\mathcal{Y}$ *on* y`)`
$y' \leftarrow$ minimum y-value in $\mathcal{Y}$
$F(P, y') \leftarrow$ coefficients A, B, C set to 0
foreach (y, R) *in* $\mathcal{Y}$ **do**
  $F(P, y) = F(P, y')$
  $i \leftarrow \{0, 1, 2, 3\}$ such that $y = h_i \in \{R.h_0, R.h_1, R.h_2, R.h_3\}$
  if $i > 0$ **then**
    $F(P, y).A \leftarrow F(P, y).A - R.h_{i-1}.A$
    $F(P, y).B \leftarrow F(P, y).B - R.h_{i-1}.B$
    $F(P, y).C \leftarrow F(P, y).C - R.h_{i-1}.C$
  $F(P, y).A \leftarrow F(P, y).A + R.h_i.A$
  $F(P, y).B \leftarrow F(P, y).B + R.h_i.B$
  $F(P, y).C \leftarrow F(P, y).C + R.h_i.C$
  $y' \leftarrow y$
$H \leftarrow$ the list of all $F(P, h)$ created above
return H

2. For δ, we perform another binary search on H to find F_δ such that $F_\delta(P, \delta)$ tells us the area of P below δ.

3. We can now calculate the area inside the query slab as $F_\delta(P, \delta) - F_\beta(P, \beta)$.

We summarise our results in the following theorem.

Theorem 6. *Let P be a monotone polygon consisting of n vertices. In $O(n \log n)$ time and $O(n)$ space, we can create a data structure that allows us to determine* $\text{area}(S \cap P)$ *in $O(\log n)$ time for any horizontal slab query region S.*

5.2 Extending to Rectangular Queries

Our actual query, as introduced in Figure 11, is a rectangular area. Our solution to this type of query is extended from the methods we used to solve a horizontal slab query.

5.3 Preprocessing

We develop a list of regions $\mathcal{R} = R_1, R_2, \ldots, R_{n-1}$ just as in Section 5.1.1. Now, instead of constructing a single list of multi-region formulas to cover all of P, we will construct a tree of such lists so that for any $1 \leq i \leq j \leq n-1$, we can query the subpolygon $\bigcup_{k=i}^{j} R_k$ for its area below a query line h in $O(\log n)$ time.

Let this multi-region formula tree be T; we construct T in the following way. First, construct the multi-region formula tree for each half of the regions recursively, giving the subtrees T_l and T_r. If $|\mathcal{R}| = 1$ for any recursive step, we create a leaf and return the area formula for that single region.

Let $H(T_l)$ and $H(T_r)$ be the multi-region formula lists for T_l and T_r, respectively. We need to create a merged list, $H(T)$ representing the regions of both subtrees together. Unfortunately, we cannot just naively merge the formulas into a combined sorted list, as each formula is only concerned with the regions upon which it was originally created.

Instead, we need to generate new coefficients for each critical height of h which will be valid for all regions of the combined area. This process is precisely Algorithm 1 *without* the initial sorting step, which we can avoid since $H(T_l)$ and $H(T_r)$ are already sorted. Thus, we can build $H(T)$ in time $O(|H(T_l)| + |H(T_r)|)$. Algorithm 2 gives more formal details.

Algorithm 2: BuildMultiRegionFormulaTree

Input: List of regions $\mathcal{R} = R_1, R_2, \ldots, R_{n-1}$

if $|\mathcal{R}| = 1$ **then**
  $T \leftarrow$ create new leaf node
  $H(T) \leftarrow F(R, h)$
  return $T, H(T)$
$m \leftarrow \left\lfloor \frac{|\mathcal{R}|}{2} \right\rfloor$
$\mathcal{R}_l = \{R_1, R_2, \ldots, R_m\}$
$\mathcal{R}_r = \{R_{m+1}, R_{m+2}, \ldots, R_{n-1}\}$
$T_l, H(T_l) \leftarrow$ `BuildMultiRegionFormulaTree(`R_l`)`
$T_r, H(T_r) \leftarrow$ `BuildMultiRegionFormulaTree(`R_r`)`
$T \leftarrow$ Create new node with left child T_l and right child T_r
$H(T) \leftarrow$ Merge $H(T_l)$ and $H(T_r)$ using Algorithm 1
return $T, H(T)$

In the last step of the algorithm we create a list of formulas over all regions, which implies that we can answer horizontal slab queries from the root of T. Recursing on half of the regions at each step gives us a tree which will be balanced with depth $O(\log n)$. At each level of T, every critical height for every region is considered, resulting in $O(n)$ area formulas. As mentioned, the merging step at each node is completed in linear time with respect to the number of regions processed. Therefore, the total time and storage required to build T

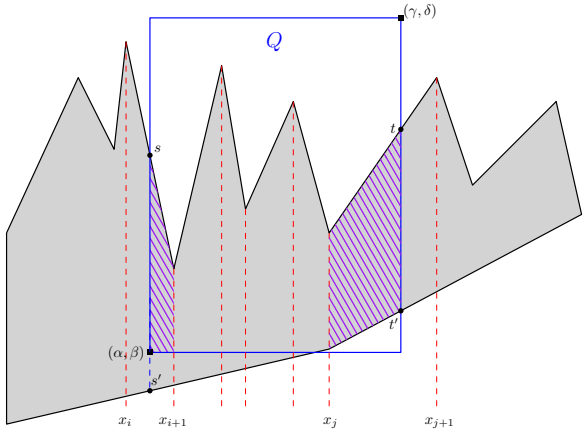


Figure 13: A monotone polygon P with query Q. The tiled areas cannot be queried using preprocessed area formulas and will require special handling.

is $O(n \log n)$.

5.3.1 Querying

Our query rectangle Q is given by the lower-left and upper-right coordinates (α, β) and (γ, δ), respectively. We define $\overline{\beta}$ and $\overline{\delta}$ as the horizontal lines through β and δ, and $\overline{\alpha}$ and $\overline{\gamma}$ as vertical lines through α and γ, respectively.

Using a binary search on the x-values of P, we can identify the regions which $\overline{\alpha}$ and $\overline{\gamma}$ pass through; let these regions be R_i and R_j, respectively. Let $V(Q)$ be the vertical slab defined by Q; that is, the vertical area between $\overline{\alpha}$ and $\overline{\gamma}$. We can calculate the area of $Q \cap P$ by considering how Q interacts with the following areas:

1. The leftmost region R_i, where $R_i \not\subset V(Q)$. Let $A_i = \text{area}(Q \cap R_i)$.
2. The center regions $R_{i+1}, R_{i+2}, \ldots, R_{j-1}$, where $R_k \subset V(Q)$ for $i+1 \leq k \leq j-1$. Let $A_c = \text{area}\Big(Q \cap \Big(\bigcup_{k=i+1}^{j-1} R_k\Big)\Big)$.
3. The rightmost region R_j, where $R_j \not\subset V(Q)$. Let $A_j = \text{area}(Q \cap R_j)$.

See Figure 13 for an example. To calculate A_i, A_c, and A_j, we begin with the center regions. All of these regions are entirely within $V(Q)$, and so we can use their precalculated area formulas directly, which is $A_c = \text{area}\Big(Q \cap \Big(\bigcup_{k=i+1}^{j-1} R_k\Big)\Big) = \sum_{k=i+1}^{j-1} F(R_k, \delta) - F(R_k, \beta)$. Performing this sum naively takes $O(n)$ time as there may be a linear number of regions spanned by Q. However, using T, we can answer this query for any values of i and j by checking at most $O(\log n)$ subtrees. At each subtree, we require $O(\log n)$ time to find the correct formula, for a total query time of $O\big(\log^2 n\big)$. However, since each subtree queries

for the same value of h, we can reduce this query time to only $O(\log n)$ by using fractional cascading [3, 4].

Considering A_i now, if $\alpha = x_i$, then $A_i = F(R_i, \delta) - F(R_i, \beta)$. In general, however, $x_i < \alpha < x_{i+1}$, and we cannot use our precalculated area formulas. Fortunately, $Q \cap R_i$ is a polygon of $O(1)$ complexity, specifically a trapezoid, so its area can be calculated directly in constant time. Recall from the construction of the list of regions, $\mathcal{R}$, that if $\overline{\alpha}$ passes between x_i and x_{i+1}, then it passes through region R_i, which stores the edges e_i and e'_i defining its top and bottom. We can calculate the intersection points with $\overline{\alpha}$ as $s = e_i \cap \overline{\alpha}$ and $s' = e'_i \cap \overline{\alpha}$. Recall that $a_{i+1} = \overline{x_{i+1}} \cap e_{i+1}$, and $a'_{i+1} = \overline{x_{i+1}} \cap e'_{i+1}$. Thus, the vertices of $Q \cap R_i$ are:

- Its top-left: the lower of (α, s) and (α, δ),
- Its bottom-left: the higher of (α, s') and (α, β),
- Its top-right: the lower of a_{i+1} and $\overline{\delta} \cap \overline{x_{i+1}}$, and
- Its bottom-right: the higher of a'_{i+1} and $\overline{\beta} \cap \overline{x_{i+1}}$.

Likewise, if $\gamma = x_j$, then $A_j = 0$. But, in general, $x_j < \gamma < x_{j+1}$, and, again, we cannot use our precalculated area formulas. $Q \cap R_j$ is also a trapezoid, however, so its area can be calculated directly in a similar way to A_i.

With all three area calculations completed, a simple sum completes our query. We summarize our results with the following theorem.

Theorem 7. *Let P be a monotone polygon consisting of n vertices. In $O(n \log n)$ time and $O(n \log n)$ space, we can create a data structure which allows us to determine* $\text{area}(Q \cap P)$ *in $O(\log n)$ time, for any axis-parallel rectangular query region Q.*

5.4 Remarks on Simple Polygons

We can apply our method for horizontal slab queries from Section 5.1 "as is" to simple polygons since nothing about the way that we decompose P, build the multi-region area formulas, or query them, depends on the monotone property. Put another way, we do not require any special handling for regions which appear above or below other regions.

To answer slab queries on simple polygons, we decompose P by extending rays from each vertex into the interior of P until they strike the next boundary. The regions will have the same 4-sided shape as in the monotone polygon case, and we can create an area formula for each one over the, at most, four critical heights of h.

Each region still maintains its own critical heights for h, and we use Algorithm 1 without modification, first sorting all critical heights together, then maintaining a set of coefficients which we adjust by region as we sweep from bottom to top.

Finally, querying the resulting list of multi-region formulas, H, is done using the same binary search method as in the monotone case. This result is summarized in the following corollary.

Corollary 7.1. *Let P be a simple polygon consisting of n vertices. In $O(n \log n)$ time and $O(n)$ space, we can create a data structure which allows us to determine* $\text{area}(S \cap P)$ *in $O(\log n)$ time for any horizontal slab query region S.*

6 Conclusion

In this paper, we introduce the *partial enclosure range searching problem*. Two variants of the problem are studied. In the first variant, a set of line segments S is preprocessed so that the partial enclosure range query for a query range Q can be performed efficiently. In the second variant, S is a polygon and Q is an axis-parallel rectangle, and the objective of the *partial enclosure area problem* is to compute the area of $S \cap Q$.

When S is a monotone polygon, our presented algorithm requires $O(n \log n)$ preprocessing time and space and the query time is $O(\log n)$. In [1], it is shown that the space can be improved to $O(n)$ by increasing the query time to $O(\sqrt{n})$. It is also shown in [1] that if S is a convex polygon then, in $O(n)$ time and space, we can create a data structure which can compute the area of $S \cap Q$ in $O(\log n)$ time, for any arbitrary oriented rectangular query range Q. For the case where S is a simple polygon, we can handle queries where Q is an axis-parallel slab. Unfortunately, we cannot extend our method for rectangular queries to work with simple polygons so easily. While the multi-region formulas themselves do not use the monotone property, our tree of multi-region formulas does. The tree functions by partitioning the trapezoidal regions with respect to vertical lines, however, in a simple polygon, a vertical line passing through the boundary of one region may pass through the interior of another. This lack of clean partitioning prevents the multi-region formula from working correctly for all possible horizontal query lines which may be given as input to the formula. Thus, the partial enclosure problem for simple polygons is worth studying.

References

[1] G. Bint. Partial enclosure range searching. Master's thesis, School of Computer Science, Carleton University, Ottawa, Canada, 2014.

[2] T. M. Chan. Optimal partition trees. *Discrete & Computational Geometry*, 47(4):661–690, 2012.

[3] B. Chazelle and L. Guibas. Fractional cascading: I. a data structuring technique. *Algorithmica*, 1:133–162, 1986.

[4] B. Chazelle and L. Guibas. Fractional cascading: II. applications. *Algorithmica*, 1:163–191, 1986.

[5] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational geometry: Algorithms and Applications*. Springer-Verlag, Berlin, third edition, 2008.

[6] R. Jarrett, G. Schobbe, M. Iwema, C. Lui, F. Jones, E. Rimas, B. Dresevic, and S. Bhattacharyay. Lasso select, Oct. 11 2011. US Patent 8,037,417.

[7] J. Matousek. Efficient partition trees. *Discrete & Computational Geometry*, 8:315–334, 1992.