Partial Enclosure Range Searching*

Gregory Bint[1] Anil Maheshwari[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 axis-parallel line segments and the goal 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 rectangle 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.

Keywords: Partial range search, simplex range searching, data sreucture, computational geometry

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*

*Research supported by NSERC.

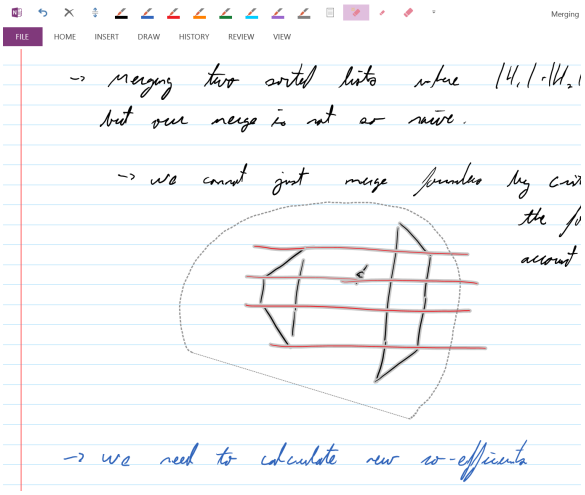

Figure 1: An example of partial enclosure range searching in Microsoft OneNote. The selected line segments are not entirely enclosed in the query region.

property if at least some fixed proportion of the object (with respect to its length, area, volume) intersects the query region Q.

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: Given a set S of line segments in $\mathbb{R}^2$, preprocess it such that for any query object Q, which is an axis-parallel rectangle or a slab bounded by two parallel lines of arbitrary orientation, the number of members in S that partially (or fully) lie in Q can be reported efficiently. In this study, we have considered S containing only axis-parallel line segments, but, Q can be a rectangle or a slab of any arbitrary orientation.

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

Table 1: Summary of Contributions

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

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*.

The paper is organized as follows. In Section 2, we state some preliminaries. Section 3 focuses on the partial enclosure range counting problem when the objects are axis-parallel 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.

Jaja et al. [7] proposed a dominance query data structure in $\mathbb{R}^d$, where the query range is of the form $[a_1,\infty]\times[a_2,\infty]\times\ldots\times[a_d,\infty]$. The preprocessing time, space and query time complexity results are $O\big(n\log^{d-2} n\big)$, $O\left(n\left(\frac{\log n}{\log\log n}\right)^{d-2}\right)$, and $O\left(\left(\frac{\log n}{\log\log n}\right)^{d-1}\right)$ respectively. Moreover, for a constant value of d, the axis-parallel hyper-rectangular range counting query can be formulated using $O(1)$ dominance query in $\mathbb{R}^d$. Thus,

Lemma 1. *Given a set P of n points in $\mathbb{R}^d$ ($d \geq 3$), it can be preprocessed in $O(n\log^{d-2} n)$ time using $O\left(n\left(\frac{\log n}{\log\log n}\right)^{d-2}\right)$ space, such that the number of points inside any given axis-parallel hyper-rectangular query region can be reported in $O\left(\left(\frac{\log n}{\log\log n}\right)^{d-1}\right)$ time.*

The range trees are used to query for an axis-parallel hyper-rectangle region in $\mathbb{R}^d$ where each side of the hyper-rectangle is a range of values of the corresponding variable. For some problems considered in this paper, the ranges may be expressed using half-planes comprising of two query variables. For those, we use the following structure described in Chan[2], restated here with $d = 2$.

Theorem 1 (Corollary 7.3 of Chan[2]). *Given a set P of n points in $\mathbb{R}^2$,*

(a) one can form $O(n)$ canonical subsets $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots\}$ (possibly overlapping) of total size $O(n \log n)$ in $O(n \log n)$ time, such that the query about the subset of all points inside any given simplex can be reported as a union of the results of disjoint canonical subsets $\{C_1, C_2, \ldots, C_k\} \subseteq \mathcal{C}$. Here, $\sum_{i=1}^{k} \sqrt{|C_i|} \leq O(\sqrt{n} \log n)$, and the overall query time is $O(\sqrt{n} \log n)$ with high probability with respect to n.

(b) for any fixed $\gamma < \frac{1}{2}$, one can form $O(n)$ (possibly overlapping) canonical subsets $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots\}$ of P of total size $O(n \log \log n)$ in $O(n \log n)$ time, such that the number of points inside any given simplex can be answered as a union of the results of disjoint canonical subsets $\{C_1, C_2, \ldots, C_k\} \subseteq \mathcal{C}$. Here, $\sum_{i=1}^{k} |C_i|^{\gamma} \leq O(\sqrt{n})$, and the overall query time is $O(\sqrt{n})$ with high probability with respect to n.

This structure is particularly well-suited for multi-part queries. To support such a query, we construct a canonical subsets structure to perform the half-plane query in the first level. With the objects of each canonical subset, we construct a secondary query structure for the second level query which has an existing query method. A similar structure with slightly worse query time bounds by Matousek[8] is presented in [5, Chapter 16] with an example of a multi-part query.

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, the secondary structure is searched to satisfy the conditions to test in the second-level. 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 object a_i also has an associated point p_i in $\mathbb{R}^2$, 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.*

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

(b) 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 $O\big(n \log^f n \log \log n\big)$

preprocessing space, $O(n \log^g n \log\log n)$ *preprocessing time, and* $O(\sqrt{n})$ *query time with high probabily.*

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

Proof. (a) Let the canonical subsets generated by Theorem 1 be $P_1, P_2, \ldots, P_m$, where $m = O(n)$, $P_i \subseteq P$, for all $i = 1, 2, \ldots, m$, and $\sum_{i=1}^m |P_i| = \sum_{i=1}^m n_i = O(n\log n)$. The time needed for this generation is $O(n\log n)$. Now, the preprocessing space required for the multi-level structure is $\sum_{i=1}^m O(n_i \log^f n_i \alpha(n_i)) = O(n\log^{f+1} n\alpha(n))$. Similarly, the preprocessing time result follows.

Query requires getting the canonical subsets in $O(\sqrt{n})$ time and the time required for searching in the associated structures of all the disjoint canonical structures $C_1, C_2, \ldots, C_k$. The total time for searching in the associated structures is $\sum_{i=1}^k Q(C_i) = \sum_{i=1}^k O(\sqrt{|C_i|}\log^h |C_i|) = \sum_{i=1}^k O(\sqrt{|C_i|}\log^h n) = O(\sqrt{n}\log^{h+1} n)$ (by Theorem 1(a)),

(b) Choosing $\gamma = \frac{1}{4}$, we have the sum of sizes of the partitions $\sum_{i=1}^m |P_i| = \sum_{i=1}^m n_i = O(n\log\log n)$, where $P_1, P_2, \ldots, P_m$ ($m = O(n)$) are the generated canonical subsets by Theorem 1(b), where $P_i \subseteq P$, for all $i = 1, 2, \ldots, m$. Thus, the preprocessing space and time of the multilevel structure are $\sum_{i=1}^m O(n_i \log^f n_i) = O(n\log^f n\log\log n)$ and $O(n\log^f n\log\log n)$ respectively. While querying, the disjoint canonical structures C_1, C_2, $\ldots$, C_k can be computed in $O(\sqrt{n})$ time, and the total time required to search in the (disjoint) C_i's is $\sum_{i=1}^k Q(C_i) = \sum_{i=1}^k O(\log^h |C_i|) \leq \sum_{i=1}^k O((|C_i|)^\gamma)$ (for any positive constant γ) $\leq O(\sqrt{n})$ (by Theorem 1(b)). Thus, the overall query time is $O(\sqrt{n})$.

(c) Here also, choosing $\gamma = \frac{1}{4}$, the sum of sizes of the partitions $\sum_{i=1}^m |P_i| = \sum_{i=1}^m n_i = O(n\log\log n)$, and the time needed for generating this partition is $O(n\log n)$. Now, the preprocessing space required for the structures associated with these partitions in the multi-level structure is $\sum_{i=1}^m O\left(n_i\left(\frac{\log n_i}{\log\log n_i}\right)^f\right) = O\left(n\left(\frac{\log n}{\log\log n}\right)^f \log\log n\right)$. Similarly, the preprocessing time result follows.

Query requires getting the canonical subsets in $O(\sqrt{n})$ time and the time required for searching in the associated structures of all the disjoint canonical structures $C_1, C_2, \ldots, C_k$. The total time for searching in the associated structures is $\sum_{i=1}^k O\left(\left(\frac{\log |C_i|}{\log\log |C_i|}\right)^h\right)$. Since $\gamma = \frac{1}{4}$ and h is a constant, we have $\left(\frac{\log |C_i|}{\log\log |C_i|}\right)^h \leq |C_i|^\gamma$. Again, as $\sum_i |C_i|^\gamma \leq O(\sqrt{n})$, we have the total query time $O(\sqrt{n})$ with high probability (see Theorem [2](b)). □

We use this corollary as a "black box" in Section 4.

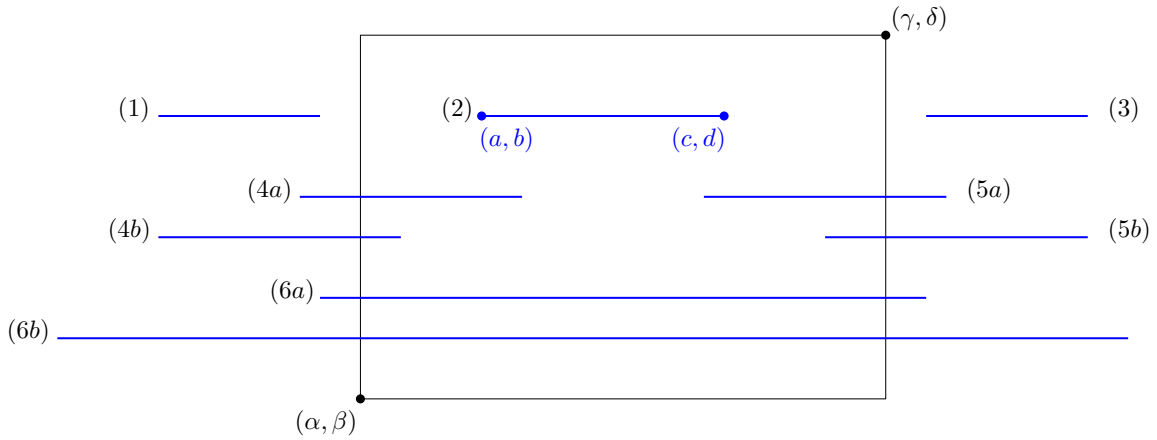

Figure 2: An axis-parallel query on axis-parallel segments. Different cases of horizontal segments interacting with the query region are shown.

3 PERS problem for Axis-Parallel Rectangles

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

Definition 1. *For a given ρ ($0 < \rho \leq 1$), 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.*

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 count all segments belonging to cases (2), $(4a)$, $(5a)$, and $(6a)$, and none of the segments belonging to any other case.

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., $S'_L = \{s_i \in S_L \mid \alpha - a_i < (1-\rho) \cdot l_i\}$. Thus, the segments in cases (4a) and (6a) satisfy this test. For segments in S_R, we want to ensure that "enough of s_i is inside Q", i.e., $S'_R = \{s_i \in S_R \mid \gamma - a_i \geq \rho l_i\}$. The cases (2) and (5a) satisfy this test. Thus, we have the following observation.

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 fourth level, at each node 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. Using Lemma 1, we have the following theorem that summarizes the solution of this problem.

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 $O\left(\left(\frac{\log n}{\log \log n}\right)^3\right)$ time, with a data structure requiring $O\left(n \left(\frac{\log n}{\log \log n}\right)^2\right)$ space created in $O\big(n \log^2 n\big)$ time in the preprocessing phase of the algorithm.*

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

Problem 2. *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 (not necessarily parallel to any one of the coordinate axis) slab Q.*

As in Section 3, 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; w is the horizontal width of Q.

Identifying whether a segment $s_i \in_\rho Q$ requires three broad steps: (i) restrict segments to those which are "not too long" to fit sufficiently inside Q, (ii) classify all segments by whether their left endpoints are left or right of L_1, and (iii) 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.

Step 1 [Restrict length]: Here, we 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 $S_1 = \{s \in S \mid \rho\ell \leq w\}$.

Step 2 [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. Note that, the segments belonging to cases (1) and (2) are only interesting to us. However, the 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 $S_R = \{s \in S_1 \mid p \text{ is right of } L_1\}$.

Step 3 [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.

Set S_L can be classified into four subsets as follows (see Figure 3(a)):

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

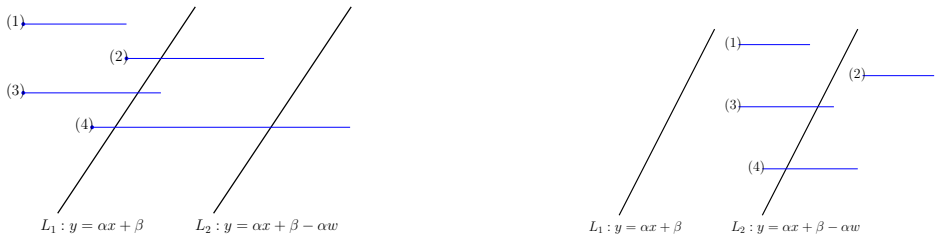

Figure 3: Segments with left endpoints (a) to the left of L_1, and (b) between L_1 and L_2.

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

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

(4) Segments which intersect both L_1 and L_2.

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

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 $y > \frac{1}{\alpha}x - \frac{\beta}{\alpha}$.

The segments in S_R can also be classified into four subsets as follows (see Figure 3(b)).

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

(2) Segments which 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 follow.

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 $y \leq \frac{1}{\alpha}x - \frac{\beta}{\alpha} + w$.

4.1.1 Data structure and complexity analysis

We will use the multi-level canonical sets data structure described in Section 2 to perform this query. Each of the three steps of the query 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 answered in $O(\log n)$ time using a 1-dimensional range tree, which can be realized as an AVL tree [5], keyed on the segment lengths. The size of this data structure is $O(n)$, and can be built in $O(n \log n)$ time during the preprocessing.

In the next level, we identify segments with partial enclosure property. We maintain two half-plane query data structures for S_L and S_R (as mentioned in the discussion after Observation 2 and 3 respectively).

With each subset created for this structure, we will associate the structure required for testing the length restriction. Applying Corollary 1.1(b) gives us the following result.

Lemma 2. *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 a specific side of a query line* and *which do not exceed a maximum length in $O(\sqrt{n})$ time, using a data structure of size $O(n \log\log n)$, which can be built in $O(n \log n \log\log n)$ preprocessing time.*

In the topmost level, we have the data structure for classifying the left-endpoints of the segments as left or right of L_1. We attach both the data structures (for S_L and S_R) at each internal node of this data structure. By Corollary 1.1(a) with $\alpha(n), \beta(n) = \log\log n$, we have the following result.

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

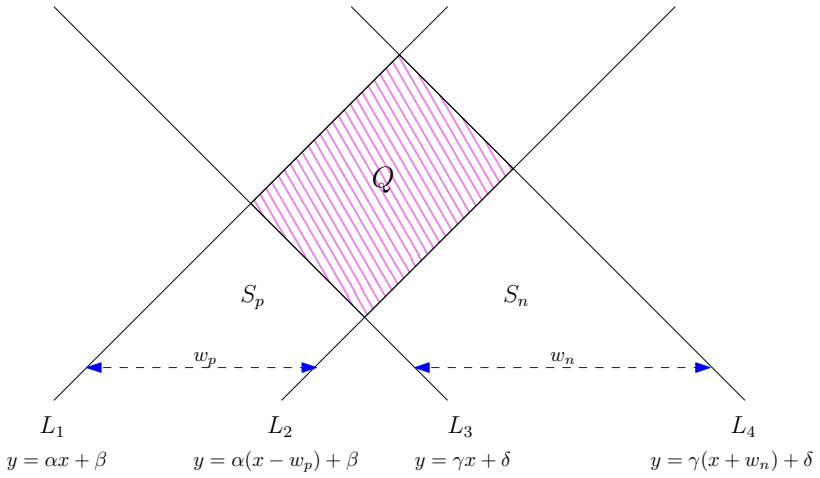

Figure 4: A query parallelogram Q formed by the inputs α, β, w_p, γ, δ, and w_n.

Finally, to fully answer the 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 3. *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 $O(\sqrt{n}\log n)$ time, using a data structure of size $O(n \log n \log\log n)$, requiring $O\big(n \log^2 n \log\log n\big)$ 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:

Problem 3. *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, $|s \cap Q| \geq \rho \cdot |s|$.*

Here the query parallelogram Q is defined by the intersection of two slabs with positive and negative slopes respectively (see Figure 4). 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. Specifically, a query region $Q = S_p \cap S_n$ is given as a 6-tuple $(\alpha, \beta, w_p, \gamma, \delta, w_n)$, where $\alpha > 0$, $w_p > 0$, $\gamma < 0$, and $w_n > 0$. More specifically, we have

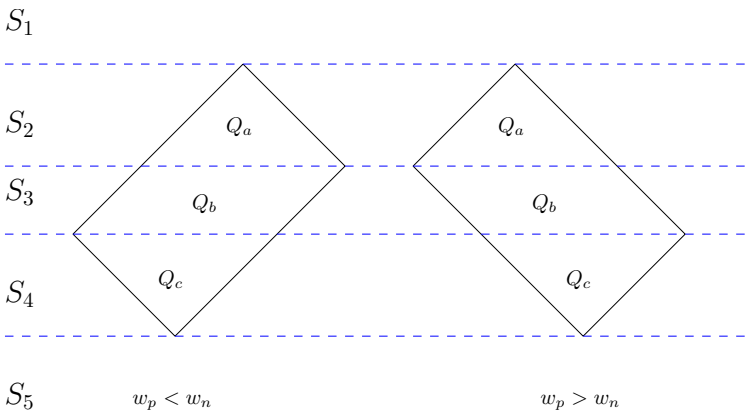

Figure 5: Decomposition of the query region Q. The orientation of Q depends on the relative widths of the slabs which define it.

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 $S_p \cap S_n$.

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.

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 defined by L_1 on the left and L_4 on the right. Likewise, Q_c, is 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 is defined by L_3 and L_4 when $w_p > w_n$. When $w_p = w_n$, Q_b disappears.

Thus, based on the query input, we can define five horizontal slabs as shown in Figure 5. The subset of segments in S falling in the i-th region is named as $S_i, i = 1, 2, 3, 4, 5$. In each horizontal slab, we have three zones, namely Z_L, Z_C, and Z_R, where the center zone Z_C is the query region itself. Figure 6 gives an illustration; Z_L, Z_C, and Z_R are coloured blue, green, and red, respectively. The segments in the i-th horizontal slab can be classified into six groups: (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), where the first and second component in the tuple of each group represents the zone containing respectively the left and right endpoint of the segments lying in that group. 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 $s \in_\rho Q$.

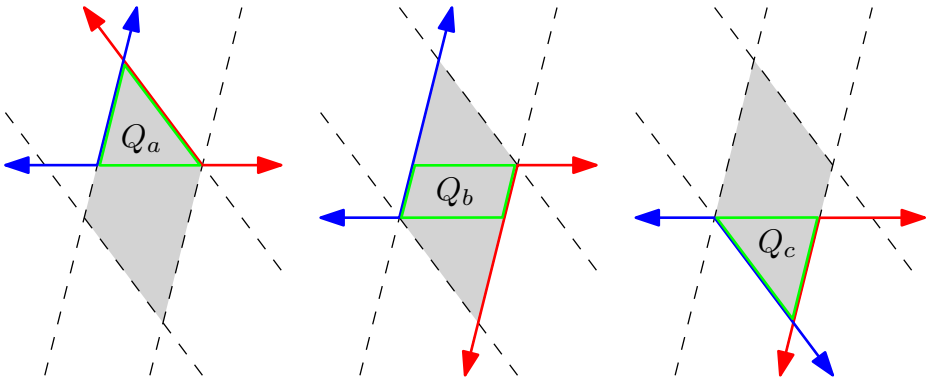

Figure 6: Classification zones for each region of Q. Each region has a left, center, and right zone, coloured blue, green, and red, respectively, where we may find segments which need further testing.

- 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.

Note that, Q_b may be a parallelogram, which we decompose into two triangular regions.

4.2.1 Partial enclosure property

We begin with examining Q_a in detail. Let $s = (a, b, \ell) \in S$ 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. We map each segment s to a 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 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:

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

We can test this expression by mapping each segment s to a 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''.

4.2.2 Construction and analysis

We will use a multi-level query structure for this problem, just as we did for the single slab query. Here, each step corresponds to one nested level of the data structure to be proposed here. Also, we explain the data structure for counting segments intersecting Q_a that satisfy the partial enclosure property. The test is decomposed in three stages.

Stage 1: In this stage, we identify the segments that lie in the horizontal slab containing Q_a slab using an AVL-tree with the y-coordinates of the left end-points $\{a_i, i = 1, 2, \ldots, n\}$ of the segments in S.

Stage 2: In this stage, we accumulate segments in each of the four groups, (Z_L, Z_C), (Z_C, Z_R), (Z_L, Z_R) and (Z_C, Z_C) by performing two nested half-plane queries for left and right end-points of the segments in S. We maintain four data structures with each node of the (primary) data structure of Stage 1 for the test of four groups, namely (Z_L, Z_C), (Z_C, Z_R), (Z_L, Z_R) and (Z_C, Z_C). The elements of S that pass the test of Stage 1 has to go through the test of all the three groups independently. All the segments in the group (Z_C, Z_C) satisfy the partial enclosure property, and this count is directly added to the result (total count of segments satisfying partial enclosure property). For other groups, we need to do the partial enclosure test in Stage 3.

Stage 3: In the next stage, for each group we identify segments satisfying the partial enclosure expression using the corresponding half-plane query data structure by applying Corollary 1.1(a),

Lemma 4. *Given a set of n horizontal line segments, we can count the subsets of S containing all the segments satisfying the partial enclosure property with respect to the triangular region Q_a obtained from the given pair of query slabs as described in Problem 3 in $O\big(\sqrt{n}\log^2 n\big)$ time, using a data structure of size $O\big(n\log^2 n\log\log n\big)$, which can be built in $O\big(n\log^3 n\log\log n\big)$ time.*

Proof. In Stage 1 the elements lying in the narrowest horizontal slab containing Q_a can be accumulated in $O(\log n)$ time using a data structure of size $O(n)$ created in $O(n\log n)$ time. All the elements in S that satisfy the test in Stage 2 can be identified in $O(\sqrt{n}\log n)$ time with preprocessing space and time $O(n\log n\log\log n)$ and $O\big(n\log^2 n\log\log n\big)$ respectively (by applying Corollary 1.1(b), and then 1.1(a)).

Finally in Stage 3, we count the segments that satisfy the partial enclosure expression among the subsets of S that satisfy the test of Stage 2. Using Corollary 1.1(a), the overall query time is $O\big(\sqrt{n}\log^2 n\big)$; the overall preprocessing space and time are $O\big(n\log^2 n\log\log n\big)$ and $O\big(n\log^3 n\log\log n\big)$ respectively. □

The same approach is followed for the other two parts Q_b and Q_c, decomposing them to triangles, if needed. Thus, we have the following result:

Theorem 4. *A set of n horizontal line segments can be preprocessed in $O\big(n\log^3 n\log\log n\big)$ time to create a data structure of size $O\big(n\log^2 n\log\log n\big)$, such that given a parallelogram Q, we can count all segments satisfying the partial enclosure property with respect to Q in $O\big(\sqrt{n}\log^2 n\big)$ time.*

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 4. *We are given a polygon P bounded by two monotone chains[1] P_L and P_U ($|P_L| + |P_U| = n$), 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.*

[1] A polygonal chain (an ordered set of line segments such that each pair of consecutive segments in that order share a common point) is said to be *monotone* if any line paralle to the y-axis cuts the chain either at one point or at no point. A polygon is said to be monotone if it is bounded by two monotone chains such that the left-most (resp. rightmost) point of both the chains are same.

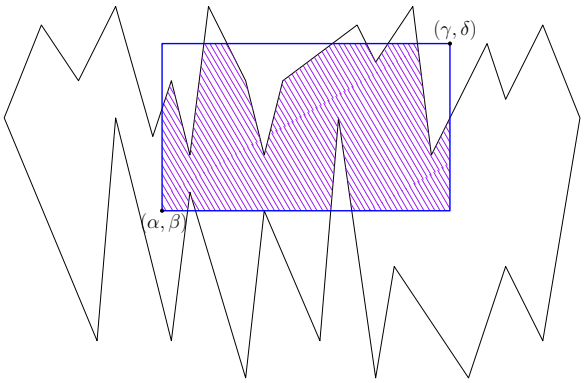

Figure 7: A monotone polygon P with query Q. The area of P enclosed by Q, $Q \cap P$, is highlighted.

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 whether the partial enclosure property is satisfied is straight-forward.

Throughout this section, we will use the following definitions. The input is a polygon P, monotone with respect to the x-axis. P is defined by its vertices $v_1, v_2, \ldots, v_n$ in clockwise order, where v_n and v_1 share an edge to close the polygon. A query rectangle Q is given by its lower-left and upper-right corners (α, β) and (γ, δ), respectively. See Figure 7 for an example.

5.1 Horizontal slab query

We will first describe an algorithm which solves the following simpler problem. Then we extend it for our main problem.

Problem 5. *Given a monotone polygon P, and a query in the form of a horizontal slab S, compute the area of P enclosed by S.*

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 formulae 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

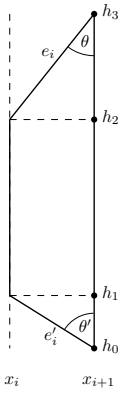

Figure 8: A trapezoidal region of P

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. Let P_U and P_L be the upper and lower chains of P, respectively. Consider the sequence of vertical 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'. If $\overline{x_i}$ 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 $R_1, R_2, \ldots, R_{n-1}$, defined as

- 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 $R_i = X_i \cap P$. Alternately, R_i is the trapezoidal region formed by the cycle $a_i, a_{i+1}, a_{i+1}', a_i'$.

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 $y = h$. Figure 8 shows an example region. From bottom to top, we have at most 4 "critical heights", where the nature of growth of a region 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 a triangle.
- h_3, where R ends.

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

$$F(R,h) = \begin{cases} 0 & \text{if } h < h_0 \\ A_1h^2 + A_2h + A_3 & \text{if } h_0 \le h < h_1 \\ B_2h + B_3 & \text{if } h_1 \le h < h_2 \\ C_1h^2 + C_2h + C_3 & \text{if } h_2 \le h < h_3 \\ D_3 & \text{if } h_3 \le h \end{cases}$$

The area functions are quadratic in h. However, for different values of h, some or all of the coefficients may be 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.

$\mathbf{h < h_0}$: The area of R below h is 0 since no part of R exists below h_0.

$\mathbf{h_0 \le h < h_1}$: Here, the area of R below the line $y = h$ grows as a triangle. Taking $h' = h - h_0$, the base of the triangle is $t' = \frac{x_{i+1}-x_i}{h_1-h_0}h'$. Thus, the area of R below h is $\frac{1}{2}h't' = A(h')^2$, where $A = \frac{1}{2}\frac{x_{i+1}-x_i}{h_1-h_0}$. Thus, refering to the area formula, we have $A_1 = A$, $A_2 = -2.A.h_0$, $A_3 = A.h_0^2$.

$\mathbf{h_1 \le h < h_2}$: Here, the area of R includes the triangular part between $y = h_0$ and $y = h_1$, and the rectangular part between $y = h_1$ and $y = h$. The area of the triangle is $C = \frac{1}{2}(x_{i+1} - x_i)(h_1 - h_0)$. Taking $h' = h - h_1$, the area of the rectangular part is Bh', where $B = (x_{i+1} - x_i)$. Thus, the total area of R is $Bh' + C$. Refering to the area formula, we have $B_2 = B$, $B_3 = C - h_1.B$.

$\mathbf{h_2 \le h < h_3}$: Here, the area of R includes the area of R includes the triangular part between $y = h_0$ and $y = h_1$, the rectangular part between $y = h_1$ and $y = h_2$, and the trapezoidal part between $y = h_2$ and $y = h$. Taking $h' = h_3 - h$, area of the trapezoidal part is $\frac{1}{2}\frac{x_{i+1}-x_i}{h_3-h_2} + ((h_3-h_2)^2 - (h')^2)$, and the rectangular part is $D = (x_{i+1} - x_i)(h_2 - h_1)$. Thus, the total area can be written as $A'(h')^2 + C' = A'(h_3 - h)^2 + C'$, where $A' = -\frac{1}{2}\frac{x_{i+1}-x_i}{h_3-h_2}$ and $C' = A + D - A'(h_3 - h_2)^2$. Refering to the area formula, we have $C_1 = -A'$, $C_2 = -2.A'.h_3$, $C_3 = C' + A'.h_3^2$.

$\mathbf{h_3 \le h}$: Here, the area of R is $D_3 = A + D + \frac{1}{2}(x_{i+1} - x_i)(h_3 - h_2)$.

5.1.3 Creating Multi-Region Formulas

We store the area formulae in an array $\mathcal{A}$. Its each entry $\mathcal{A}[i]$ corresponds to an interval $[x_i, x_{i+1}]$, $i = 0, 1, 2 \ldots, n-1$, and it consists of 13 fields, namely $h_0, h_1, h_2, h_3, A_1, A_2, A_3, B_2, B_3, C_1, C_2, C_3, D_3$, as mentioned in the expression of the area formula in the earlier section. We create another array $\mathcal{B}$, whose entries correspond to the y-coordinates of the n vertices of P, and each entry consists of four fields, namely h, Q, L and C, which stand for the coefficient of quadratic, linear and constant terms in the overall area expression below a

given horizontal line $y = h$. Initially, we assume that the y-coordinates of the vertices of P are distinct. We maintain four lists $H_j, j = 0, 1, 2, 3$, where H_j contains h_j values for all the n intervals $(x_i, x_{i+1}], i = 0, 1, \ldots, n-1$. Each element contains its corresponding vertex number in P.

We sort the vertices of P in increasing order of their y-coordinates, and process them in order. While processing an element (h, k) (where k is the vertex number corresponding to h), h lies in the $k-1$-th and k-th element in the array $\mathcal{A}$. Suppose $h = \mathcal{A}[k].h_j$ (h_j-th field of $\mathcal{A}[k]$), $j \in \{0, 1, 2, 3\}$. We update Q, L and C by deleting $j-1$-th area expression of $\mathcal{A}[k-1]$ and $\mathcal{A}[k]$ and adding j-th area expression of $\mathcal{A}[k-1]$ and $\mathcal{A}[k]$, and store them in the Q, L and C fields of $\mathcal{B}[h]$.

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

5.1.4 Querying Multi-Region Formulas

For an input height y of a query, we need to report the area of the portion of the polygon P below y. We can answer this query by searching y among the h-field of the elements in $\mathcal{B}$, and then computing $AREA = Q[\alpha].y^2 + L[\alpha].y + C[\alpha]$, where $y \in [\mathcal{B}[\alpha - 1].h, \mathcal{B}[\alpha].h]$.

We summarise our results in the following theorem.

Theorem 5. *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 7, 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.

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 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

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 $T, H(T)$

$\overline{\gamma}$ 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 9 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

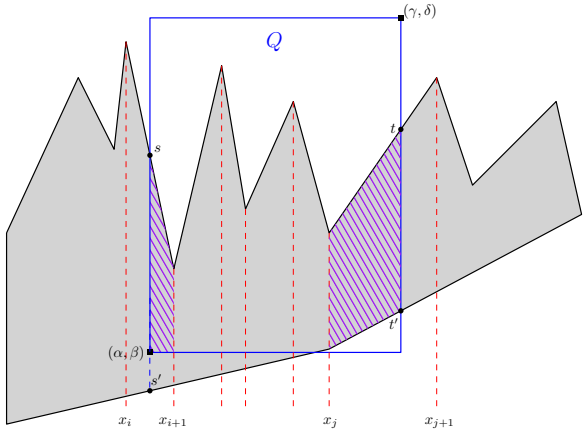

Figure 9: A monotone polygon P with query Q. The tiled areas cannot be queried using preprocessed area formulas and will require special handling.

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

Theorem 6. *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* 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 "as it is" to simple polygons. We compute the trapezoidal decomposition of the given (simple) polygon P, and store the trapezoids in an array. For each trapezoid R, we have a a multi-region area formulas $F(R, h)$ which is a function of the height h (see Section 5.1.2). Using four priority queue data structures storing the four critical heights of the generated trapezoids, we can implement a horizontal line sweep to compute the fields $Q(h)$, $L(h)$ and $C(h)$ for each critical height $h \in H$, where H is the set of critical heights of all the trapezoids. Now, the query about the area inside a horizontal slab S is answered by performing binary search as in Section 5.1.4. The result is summarized in the following corollary.

Corollary 6.1. *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* 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 $S \cap Q$.

In the first variant, we have proposed query algorithms (i) when the given objects S are axis-parallel line segments, and Q to be an axis-parallel rectangle, or a slab of arbitrary orientation, or the intersection of two slabs of arbitrary orientation.

In the second variant, 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. Jaja, C. W. Mortensen and Q. Shi. Space-Efficient and Fast Algorithms for Multidimensional Dominance Reporting and Counting. Proc. *ISAAC 2004*, pages 558-568.

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