A Linear Time Algorithm for the Maximum Overlap of Two Convex Polygons Under Translation

Timothy M. Chan ✉ 📵
Siebel School of Computing and Data Science, University of Illinois at Urbana-Champaign, USA

Isaac M. Hair ✉ 📵
Department of Computer Science, University of California, Santa Barbara, USA

──── **Abstract** ────

Given two convex polygons P and Q with n and m edges, the *maximum overlap* problem is to find a translation of P that maximizes the area of its intersection with Q. We give the first randomized algorithm for this problem with linear running time. Our result improves the previous two-and-a-half-decades-old algorithm by de Berg, Cheong, Devillers, van Kreveld, and Teillaud (1998), which ran in $O((n + m) \log(n + m))$ time, as well as multiple recent algorithms given for special cases of the problem.

1 Introduction

Problems related to convex polygons are widely studied in computational geometry, as they are fundamental and give insights to more complex problems. We consider one of the most basic problems in this class:

▶ **Problem 1.** *Given two convex polygons P and Q in the plane, with n and m edges respectively, find a vector $t \in \mathbb{R}^2$ that maximizes the area of $(P + t) \cap Q$, where $P + t$ denotes P translated by t.*

This problem for convex polygons was first explicitly posed as an open question by Mount, Silverman, and Wu [32]. In 1998, de Berg, Cheong, Devillers, van Kreveld, and Teillaud [20] presented the first efficient algorithm which solves the problem in $O((n + m) \log(n + m))$ time. As many problems have $n \log n$ complexity, this would appear to be the end of the story for the above problem—or is it?

Motivation and Background. The computational geometry literature is replete with various types of problems about polygons [34], e.g., polygon containment [14] (which has seen exciting development even in the convex polygon case, with translations and rotations, as recently as last year's SoCG [13]). The maximum overlap problem may be viewed as an outgrowth of polygon containment problems.

The maximum overlap problem may also be viewed as a formulation of *shape matching* [7], which has numerous applications and has long been a popular topic in computational geometry. Many notions of distance between shapes have been used in the past (e.g., Hausdorff distance, Fréchet distance, etc.), and the overlap area is another very natural measure (the larger it is, the closer the two shapes are) and it has been considered in numerous papers for different classes of objects (e.g., two convex polytopes in higher dimensions [3, 5, 4], two arbitrary polygons in the plane [18, 24], two unions of balls [10], one convex polygon vs. a discrete

point set[1] [9, 1], etc.) as well as different types of motions allowed (translations, rotations, and scaling—with translations-only being the most often studied). Problem 1 is perhaps the simplest and most basic version, but is already quite fascinating from the theoretical or technical perspective, as we will see.

A Superlinear Barrier?   Due to the concavity of (the square root of) the objective function [20], it is not difficult to obtain an $O((n + m) \operatorname{polylog}(n + m))$-time algorithm for the problem by applying the well-known *parametric search* technique [28, 2] (or more precisely, a multidimensional version of parametric search [35]). De Berg, Cheong, Deviller, van Kreveld, and Teillaud [20] took more care in eliminating extra logarithmic factors to obtain their $O((n + m) \log(n + m))$-time algorithm, by avoiding parametric search and instead using more elementary binary search and matrix search techniques [23]. But any form of binary search would seem to generate at least one logarithmic factor, and so it is tempting to believe that their time bound might be the best possible.

On the other hand, in the geometric optimization literature, there is another well-known technique that can give rise to *linear*-time algorithms for various problems: namely, *prune-and-search* (pioneered by Megiddo and Dyer [29, 21, 30]). For prune-and-search to be applicable, one needs a way to prune a fraction of the input elements at each iteration, but our problem is sufficiently complex that it is unclear how one could throw away any vertex from the input polygons and preserve the answer.

Perhaps because of these reasons, no improved algorithms have been reported since de Berg et al.'s 1998 work. This is not because of lack of interest in finding faster algorithms. For example, a (lesser known) paper by Kim, Choi, and Ahn [27] considered an unbalanced case of Problem 1 and gave an algorithm with $O(n + m^2 \log^3 n)$ running time, which is linear when $m \ll \sqrt{n / \log^3 n}$. Ahn et al. [6] investigated *approximation* algorithms for the problem and obtain logarithmic running time for approximation factor arbitrarily close to 1, while Har-Peled and Roy [24] more generally gave a linear-time algorithm to approximate the entire objective function, which was then used to obtain a linear-time approximation algorithm for an extension of the problem to polygons that are decomposable into $O(1)$ convex pieces.

Our Result.   We obtain a randomized (exact) algorithm for Problem 1, running in *linear* (i.e., $O(n + m)$) expected time! This is the first improvement to de Berg et al.'s result [20] in 26 years, and is clearly optimal, and so fully settles the complexity of Problem 1.

A New Kind of Prune-and-Search?   Besides our result itself, the way we obtain linear time is also interesting. As mentioned, if one doesn't care about extra $\log(n + m)$ factors, one can use (multidimensional) parametric search to solve the problem. Specifically, parametric search allows us to reduce the optimization problem to the implementation of a *line oracle* (determining which side of a given line the optimal point t^* lies in), which in turn can be reduced to a *point oracle* (evaluating the area of $(P + t) \cap Q$ for a given point t). A point oracle can be implemented in linear time by computing the intersection $(P + t) \cap Q$ explicitly. (In de Berg et al.'s work [20], they directly implemented the line oracle in linear time.)

Various linear-time prune-and-search algorithms (e.g., for low-dimensional linear programming [29, 21, 30], ham-sandwich cuts [31], centerpoints [25], Euclidean 1-center [22], rectilinear 1-median [36, 33], etc.) also exploit line or point oracles; using (in modern

---

[1]   In the discrete case, we are maximizing the number of points inside the translated polygon.

parlance) *cuttings* [16], such oracle calls let them refine the search space and eliminate a fraction of the input at each iteration. Unfortunately, for our problem, we can't technically eliminate any part of the input polygons P and Q. Instead, our new idea is to reduce the *cost* of the oracles iteratively as the search space is refined, which effectively is as good as reducing the input complexity itself. As the algorithm progresses, oracles get cheaper and cheaper, and we will bound the total running time by a super-geometric series.

To execute this strategy, one should think of oracles as data structures with sublinear query time (i.e., sublinear in the original input size $n + m$). Our key idea to designing such data structures is to divide the input polygons into *blocks* of a certain size based on angles/slopes. This idea is inspired by the recent work by Chan and Hair [13] (they studied the convex polygon containment problem with both translations and rotations, and although our problem without rotations might seem quite different, their divide-and-conquer algorithms also crucially relied on division of the input convex polygons by angles/slopes). We show that at each iteration, as we refine the search space, we can use the current "block data structure" to obtain a better "block data structure" with a larger block size. (The refinement process from one block structure to another block structure shares some general similarities with the idea of *fractional cascading* [17].)

We are not aware of any prior algorithms in the computational geometry literature that work in the same way (which makes our techniques exciting). A linear-time algorithm by Chan [12] for matrix the searching problem (reporting all elements in a Monge matrix less than given value) has a recurrence that looks similar to the one we obtain here, but this appears to be a coincidence. Chazelle's infamous linear-time algorithm for triangulating simple polygons [15] also iteratively refines a "granularity" parameter analogous to our block size, and as one of many steps, also requires the implementation of oracles (for ray shooting, in his case) whose cost similarly varies as a function of the granularity (at least from a superficial reading of his paper)—fortunately, our algorithm here will not be as complicated!

Remarks. Very recently, Jung, Kang, and Ahn [26] have announced a *linear*-time algorithm for a related problem: given two convex polygons P and Q in the plane, find a vector $t \in \mathbb{R}^2$ that minimizes the area of the convex hull $(P + t) \cup Q$. Although the problem may look similar to our problem on the surface, it is in many ways different: in the minimum convex hull problem, the optimal point t^* is known to lie on one of $O(n + m)$ candidate lines (formed by an edge of one polygon and a corresponding extreme vertex of the other polygon), whereas in the maximum overlap problem (Problem 1), t^* lies on one of $O((n + m)^2)$ candidate lines (defined by pairs of edges of the two polygons). Thus a more traditional prune-and-search approach can be used to solve the minimum convex hull problem, but not the maximum overlap problem. Indeed, Jung et al.'s paper [26] considered the maximum overlap problem, but they do not give an improvement over de Berg et al.'s $O((n + m) \log(n + m))$ time algorithm for the two dimensional case. Instead, they give some new results for the problem in higher dimensions d, but their time bound is $O(n^{2\lfloor d/2 \rfloor})$, which is much larger than linear.

We observe that, in the specific case of maximum overlap for convex polytopes in dimension $d = 3$, one can obtain a significantly faster $O((n + m) \operatorname{polylog}(n + m))$-time algorithm by directly using multi-dimensional parametric search [35], since a point oracle (computing the area of the intersection of two convex polytopes in $\mathbb{R}^3$) can be done in $O(n + m)$ time by explicitly computing the intersection [15, 11], and this is parallelizable with $O((n + m) \operatorname{polylog}(n + m))$ work and $O(\operatorname{polylog}(n + m))$ steps by known parallel algorithms for intersection of halfspaces in $\mathbb{R}^3$, i.e., convex hull in $\mathbb{R}^3$ in the dual (e.g., see [8]). This observation seems to have not appeared in literature before (including [37, 26]).

2 Preliminaries

We may assume that P and Q have no (adjacent) parallel edges, as these can be merged. Without loss of generality, P and Q both contain the origin as an interior point. Throughout, we assume that the edges of P and Q are given in counterclockwise order. We define $N := n + m$, and we use $[x]$ to denote the set $\{1, \ldots, x\}$. If x is not an interger, then we use $[x]$ to denote the set $\{1, \ldots, \lceil x \rceil\}$, i.e. x is implicitly rounded up to the nearest integer. "*With high probability*" means "with probability at least $1 - n^{-\Omega(1)}$." The *interior* of a triangle refers to all points strictly inside of its bounding edges, and the *interior* of a line segment refers to all points strictly between its bounding points.

The Objective Function.   We want to find a translation maximizing

$$\text{Area}((P + t) \cap Q).$$

De Berg, Cheong, Devillers, van Kreveld, and Teillaud show several interesting properties of the area as a function of t [20]. For our purposes, we only need to import the following.

▶ **Lemma 1.** $\sqrt{\text{Area}((P + t) \cap Q)}$ *is downward concave over all values of t that yield nonzero overlap of $(P + t)$ with Q.*

The restriction on t is important, as $A(t)$ suddenly transitions from a convex function to a constant function $A(t) = 0$ when t is outside of the specified region. For our algorithm, however, this technicality will have little impact.

Cuttings.   First, we give a definition of the type of cuttings we use.

▶ **Definition 2** (ε-Cuttings)**.** *A cutting for a set X of n hyperplanes in $\mathbb{R}^d$, for any $d \geq 1$, is a partition of $\mathbb{R}^d$ into simplices such that the interior of every simplex intersects at most $\varepsilon \cdot n$ hyperplanes of X.*

Endowed with the ability to sample random hyperplanes, we can actually produce such a cutting with very high probability in sublinear time, as per Clarkson and Shor [19].

▶ **Lemma 3.** *Let X be a set of n hyperplanes in $\mathbb{R}^d$, for any $d \geq 1$. Assume we can sample a uniformly random member of X in expected time $O(n^{0.1})$. Let ε be any constant. Then in $O(n^{0.2})$ expected time, we can find a set of $O(1)$ simplices that constitutes an ε-cutting of X with high probability.*

Area Prefix Sums.   Let $v_1, \ldots v_n$ be the vertices of P listed in counterclockwise order, and let $v'_1, \ldots v'_m$ be the vertices of Q listed in counterclockwise order. $P[v_x : v_y]$ denotes the convex hull of the set of points containing: (i) the origin, (ii) the vertices v_x and v_y, and (iii) all vertices encountered when traversing the boundary of P in counterclockwise order from v_x to v_y. We define $Q[v'_x : v'_y]$ analogously.

By a standard application of prefix sums, we can produce a data structure to report $\text{Area}(P[v_x : v_y])$ and $\text{Area}(Q[v'_x : v'_y])$ with $O(N)$ preprocessing time and $O(1)$ query time. This is since (taking $\text{Area}(P[v_x : v_y])$ as an example):

$$\text{Area}(P[v_x : v_y]) = \text{Area}(P[v_1 : v_y]) - \text{Area}(P[v_1 : v_x]) \text{ when } y > x, \text{ and}$$
$$\text{Area}(P[v_x : v_y]) = \text{Area}(P) - (\text{Area}(P[v_1 : v_x]) - \text{Area}(P[v_1 : v_y])) \text{ when } y < x.$$

Throughout, we refer to the above data structure as *the area prefix sums for P and Q*.

The Configuration Space. Consider two convex polygons A and B, or more generally, two polygonal chains A and B. Let e be an edge of A and let e' be an edge of B. These two edges define a parallelogram $\pi(e, e')$ in configuration space such that, for any vector $t \in \mathbb{R}^2$, we have that $e + t$ intersects e' iff $t \in \pi(e, e')$. Let $\partial\pi(e, e')$ denote the set of four line segments defining $\pi(e, e')$. We define the set of all such line segments as

$$\Pi(A, B) := \bigcup_{\text{edge } e \text{ of } A, \text{ edge } e' \text{ of } B} \partial\pi(e, e'). \tag{1}$$

We define $\overleftrightarrow{\Pi}(A, B)$ to be $\Pi(A, B)$ where each line segment is extended to a line.

Angles and Angle Ranges. The *angle* of an edge e on the boundary of an arbitrary convex polygon A, denoted $\theta_A(e)$, is the angle (measured counterclockwise) starting from a horizontal rightward ray and ending with a ray coincident with e that has A on its left side. We always have that $\theta_A(e) \in [0, 2\pi)$. For any subset X of the edges of A, let $\Lambda_A(X)$ denote the minimal interval in $[0, 2\pi)$ such that $\theta_A(e) \in \Lambda_A(X)$ for all $e \in X$. We call $\Lambda_A(X)$ the *angle range* for the subset X with respect to A.

3 Blocking Scheme

Given a parameter $b \in \mathbb{N}$, referred to as the *block size*, we define a partition[2] of P and Q into *blocks* $P_1, \ldots P_{\lceil N/b \rceil}$ and $Q_1, \ldots Q_{\lceil N/b \rceil}$ as follows. First (implicitly) sort the set[3]

$$\{\theta_P(e) : e \text{ is an edge of } P\} \cup \{\theta_Q(e) : e \text{ is an edge of } Q\}$$

into increasing order and then partition the resulting sequence into consecutive subsequences $S_1, \ldots S_{\lceil N/b \rceil}$ of length b each (except possibly the last one). We then define P_i as the convex hull of the origin and the set of edges[4] $\{e : e \text{ is an edge of } P, \text{ and } \theta_P(e) \in S_i\}$. Q_j is defined similarly. See Figure 1 for an example.

It turns out that b uniquely defines the partition, so we refer to $P_1, \ldots P_{\lceil N/b \rceil}$ and $Q_1, \ldots Q_{\lceil N/b \rceil}$ as the *blocks determined by parameter b*. This scheme is useful because the angle ranges $\Lambda_P(E_P(P_i))$ and $\Lambda_Q(E_Q(Q_j))$ are strictly disjoint whenever $i \neq j$, where $E_P(P_i)$ is the set of edges that P_i shares with P, and $E_Q(Q_j)$ is the set of edges that Q_j shares with Q. We call $E_P(P_i)$ the *restriction of P_i's edges to P*, and similarly for $E_Q(Q_j)$.

3.1 A Data Structure Using Blocks

Towards producing a prune-and-search style algorithm, we want some way to reduce the time required to compute the vector t^* which maximizes $\text{Area}((P + t^*) \cap Q)$. Instead of reducing the complexity of P or Q directly, we maintain and refine a specific data structure that contains information about P and Q and allows us to rapidly search for optimal placements.

▶ **Definition 4** (($\mu, b, \mathcal{T}$)-Block Structure). *A data structure containing the following:*

1. *Access to the vertices of P and Q in counterclockwise order with $O(1)$ query time, and access to the area prefix sums for P and Q with $O(1)$ query time.*

---

[2] Actually, it isn't quite a partition because the boundaries may overlap, but these have zero area.
[3] We assume that no angles are shared between the edges of P and Q, as this case can be handled easily but requires a clumsier definition.
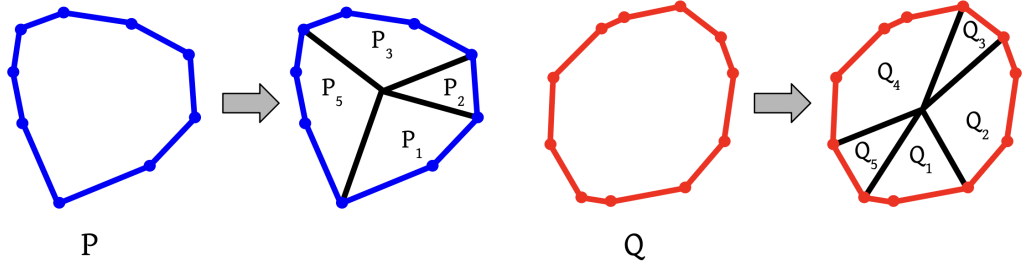[4] The set of edges may be empty, in which case the block is just a single point (the origin).

Figure 1 A partition of example polygons P and Q into blocks, with $b = 4$. Note that the spatial *sizes* of blocks may vary greatly, and the number of edges in each P_i and Q_j may differ. In fact, for some indices i, one of P_i or Q_j may contain just the origin.

2. *A block size parameter b, which implies blocks $P_1, \ldots, P_{\lceil N/b \rceil}$ and $Q_1, \ldots, Q_{\lceil N/b \rceil}$.*
3. *A region $\mathcal{T} \subset \mathbb{R}^2$ that is either (i) a triangle, (ii) a line segment, or (iii) a point.*
4. *A partition of $S := [N/b]$ into subsets S_{Good} and S_{Bad}, with the requirement $|S_{\text{Bad}}| \leq \mu$.*
5. *For every $i \in S_{\text{Good}}$, access in time $O(1)$ to a constant complexity quadratic function f_i such that $f_i(t) = \text{Area}((P_i + t) \cap Q_i)$ for all $t \in \mathcal{T}$.*

Throughout, we assume that S_{Good} and S_{Bad} are sorted, since we can apply radix sort in time $O(|S_{\text{Good}}| + |S_{\text{Bad}}|)$. When using the block structure, we aim to *decrease* the size of $\mathcal{T}$ while *increasing* the block size b. If this is done carefully enough, we get a surprising result: a sublinear time oracle to report $\text{Area}((P + t) \cap Q)$ for any translation $t \in \mathcal{T}$.

4 The Linear Time Algorithm

We will reduce Problem 1 to a few different subroutines/oracles. Roughly speaking, these allow us to produce a highly refined block structure in linear time, which can be used along with multidimensional parametric search [35] to find the translation of P maximizing its area of intersection with Q in time $O(N/ \log N)$.

▶ **Problem 2** (MAXREGION). *Given a $(\mu, b, \mathcal{T})$-block structure for P and Q, find*

$$\max_{t \in \mathcal{T}} (\text{Area}((P + t) \cap Q))$$

along with the translation $t^* \in \mathbb{R}^2$ realizing this maximum.[5]

▶ **Problem 3** (CASCADE). *Given a $(\mu, b, \mathcal{T})$-block structure, parameters μ' and b' such that b divides b', and a triangle or line segment $\mathcal{T}' \subseteq \mathcal{T}$, either (i) produce a $(\mu', b', \mathcal{T}')$-block structure, or (ii) report that the interior of $\mathcal{T}'$ intersects more than $\mu'/2$ lines from the set*

$$\mathcal{S} := \bigcup_{k \in \{0, \ldots, \lceil N/b' \rceil - 1\}} \left(\bigcup_{(i,j) \in [\frac{b'}{b}] \times [\frac{b'}{b}]} \overleftrightarrow{\Pi}(P_{k \cdot \frac{b'}{b} + i}, Q_{k \cdot \frac{b'}{b} + j}) \right), \tag{2}$$

where $P_1, \ldots, P_{\lceil N/b \rceil}$ and $Q_1, \ldots, Q_{\lceil N/b \rceil}$ are the blocks of P and Q determined by b.

---

[5] During some calls to MAXREGION, $\mathcal{T}$ may *not* contain the global optimum.

The set $\mathcal{S}$ may appear mysterious, but it simply corresponds to the line extensions for the edges that could "cause trouble" for the cascading procedure. The only observation necessary right now is that $|\mathcal{S}| = O(Nb')$, i.e. $\mathcal{S}$ is near-linear in size for all $b' = N^{o(1)}$.

We write the expected time complexity of the fastest algorithm for MaxRegion as $T_{d\text{-Max}}(N, b, \mu)$, where d is the dimension of the region $\mathcal{T}$. We write the expected time complexity of the fastest algorithm for Cascade as $T_{\text{Cascade}}(N, b, b', \mu)$. The latter will not depend on μ' or the dimensions of $\mathcal{T}$ and $\mathcal{T}'$.

The next lemmas will be proved in Section 5 and Section 6. The main idea behind proving each is to group most of the blocks of P and Q into larger convex polygons that intersect in at most a constant number of locations, and then to process the rest via brute-force.

▶ **Lemma 5** (Point Oracle). *For all $2 \le b \le N$, for all μ, we have:*

$$T_{0\text{-}Max}(N, b, \mu) = O(N \cdot \frac{\log^2 b}{b} + \mu b).$$

▶ **Lemma 6** (Cascading Subroutine). *For all $2 \le b, b' \le N$ such that b divides b', for all μ, we have:*

$$T_{Cascade}(N, b, b', \mu) = O(N \cdot \frac{\log b' \log b}{b} + \mu b^2).$$

4.1 Reducing to the Point Oracle and Cascading Subroutine

We now give an algorithm for MaxRegion when the input $(\mu, b, \mathcal{T})$-block structure has $\mathcal{T}$ being a triangle or line segment. The algorithm works by invoking MaxRegion on regions with lower dimension to iteratively shrink $\mathcal{T}$.

▶ **Lemma 7.** *For all $2 \le b, b' \le N^{o(1)}$ such that b divides b', for all μ, μ' such that $\mu' = N^{1-o(1)}$, we have:*

$$T_{2\text{-}Max}(N, b, \mu) = O(\log(Nb'/\mu')) \cdot T_{1\text{-}Max}(N, b, \mu) + O(N^{0.2+o(1)}) +$$
$$O(1) \cdot T_{Cascade}(N, b, b', \mu) + T_{2\text{-}Max}(N, b', \mu')$$

$$T_{1\text{-}Max}(N, b, \mu) = O(\log(Nb'/\mu')) \cdot T_{0\text{-}Max}(N, b, \mu) + O(N^{0.2+o(1)}) +$$
$$O(1) \cdot T_{Cascade}(N, b, b', \mu) + T_{1\text{-}Max}(N, b', \mu')$$

Proof. The algorithm for when $\mathcal{T}$ is a line segment is nearly identical to the algorithm for when $\mathcal{T}$ is a triangle, so for brevity we only focus on the triangle case. Let $P_1, \ldots P_{\lceil N/b \rceil}$ and $Q_1, \ldots Q_{\lceil N/b \rceil}$ be the blocks of P and Q determined by block size parameter b.

Define $\mathcal{T}_0 := \mathcal{T}$. We will produce a series of triangles $\mathcal{T}_0 \supset \mathcal{T}_1 \supset \mathcal{T}_2 \supset \ldots$ such that each triangle is guaranteed to contain the optimal translation t^* for the original region $\mathcal{T}$, and then take the final triangle in this sequence as $\mathcal{T}'$. Once we have $\mathcal{T}'$, we invoke Cascade to produce a $(\mu', b', \mathcal{T}')$-block structure, and then find the optimal translation within $\mathcal{T}'$ (along with the area for this translation) by invoking MaxRegion on this new block structure.

In the subsequent analysis, we show that, with high probability, the interior of $\mathcal{T}'$ produced as per our sequence will intersect at most $\mu'/2$ lines of the set $\mathcal{S}$. If this event occurs, then Cascade will be *forced* to produce a $(\mu', b', \mathcal{T}')$-block structure. If this event does not occur, and Cascade returns failure, we can re-run the randomized algorithm to produce a new $\mathcal{T}'$ and try again. The expected number of attempts is $O(1)$, and so all of the steps (except for producing $\mathcal{T}'$) have a time overhead of $O(1) \cdot T_{\text{Cascade}}(N, b, b', \mu) + T_{2\text{-Max}}(N, b', \mu')$.
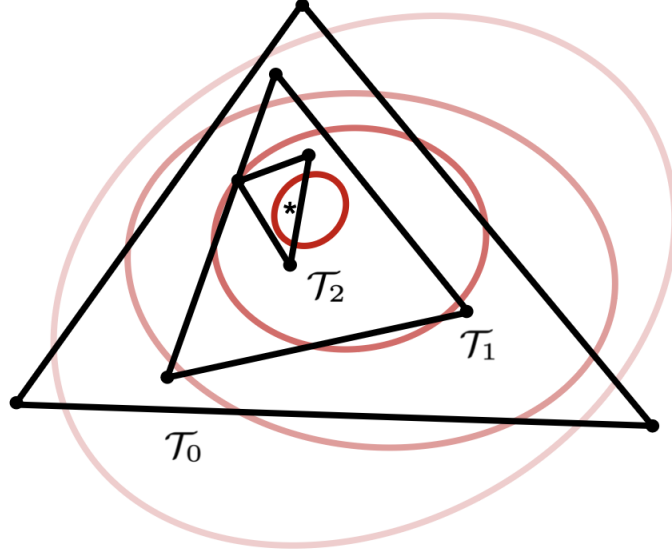
Figure 2 A series of triangles $\mathcal{T}_0$, $\mathcal{T}_1$, and $\mathcal{T}_2$ as produced by the algorithm. The (unknown) optimal placement t^* is denoted by an asterisk. The red curves indicate contour lines of the objective function, with darker circles indicating larger values of Area$((P + t) \cap Q)$.

Producing $\mathcal{T}'$. As per the above, the expected number of times we need to make $\mathcal{T}'$ is $O(1)$, so we only focus on the time complexity to make $\mathcal{T}'$ once. Consider producing a triangle $\mathcal{T}_{x+1}$ from a triangle $\mathcal{T}_x$ in the sequence. As a first step, we use Lemma 3 to make, with high probability, a $\frac{1}{2}$-cutting for the lines of $\mathcal{S}$ that intersect the interior of $\mathcal{T}_x$.

To apply this lemma, we need an efficient sampler. We argue that naive rejection sampling will suffice. Observe that if rejection sampling fails to find a random line of $\mathcal{S}$ that intersects the interior of $\mathcal{T}_x$ within time $O(n^{0.1})$, then with high probability only $N^{1-\Omega(1)}$ lines of $\mathcal{S}$ intersect $\mathcal{T}_x$. By assumption $\mu'/2 = N^{1-o(1)}$, so we can directly take $\mathcal{T}' = \mathcal{T}_x$. If rejection sampling is fast enough, then we can produce the required $\frac{1}{2}$-cutting with high probability in time $O(|\mathcal{S}|^{0.2}) = O(N^{0.2+o(1)})$. Then, we intersect this cutting with $\mathcal{T}_x$ and re-triangulate the resulting cells, which takes $O(1)$ time in total.

The final step is to locate the cell of this triangulation that contains t^*. This cell will be $\mathcal{T}_{x+1}$. To do this, we check all $O(1)$ triangles by recursing in the dimension. We assume that t^* is strictly contained within some cell, as the case that it is on the boundary can be detected similarly. Consider a single triangle $\mathcal{X}$. Let $\mathcal{X}^-$ be an infinitesimally smaller copy of $\mathcal{X}$ that is strictly inscribed within $\mathcal{X}$.[6] All $O(1)$ bounding line segments for $\mathcal{X}$ and $\mathcal{X}^-$ are contained within $\mathcal{T}$. Therefore, we can use the input $(\mu, b, \mathcal{T})$-block structure as a block structure for these line segments, and find the optimal translation (along with the area of intersection) restricted to each line segment in time $O(1) \cdot T_{1\text{-Max}}(N, b, \mu)$. If one of the line segments for $\mathcal{X}^-$ has a translation realizing a larger area than all of the line segments for $\mathcal{X}$, then by Lemma 1 we know that $t^* \in \mathcal{X}$. Otherwise, $\mathcal{X}$ does not contain t^*.[7]

---

[6] Infinitesimal shifts are a standard trick, and all of the algorithms in this paper can handle them without modification as long as they are implemented carefully enough.

[7] All invocations might return a maximum area of zero, but we can handle this in $O(1)$ time by checking whether an arbitrary translation that causes P and Q to overlap is located within $\mathcal{X}$.

With high probability, the interior of $\mathcal{T}_{x+1}$ will intersect at most half as many lines of $\mathcal{S}$ as $\mathcal{T}_x$. By construction and the fact that b divides b' (and hence $b \leq b'$), we have $|\mathcal{S}| = O(Nb')$. Thus there are at most $O(\log \frac{|\mathcal{S}|}{\mu'}) = O(\log(Nb'/\mu'))$ iterations before we produce a triangle that intersects at most $\mu'/2$ lines of $\mathcal{S}$ with high probability and hence can be taken as $\mathcal{T}'$. The time for all iterations is $O(\log(Nb'/\mu')) \cdot (T_{\text{1-Max}}(N, b, \mu) + O(N^{0.2+o(1)})) = O(\log(Nb'/\mu')) \cdot T_{\text{1-Max}}(N, b, \mu) + O(N^{0.2+o(1)})$. ◀

4.2 Putting it all Together

We now present our linear time algorithm. This amounts to solving a curious recurrence, and we actually have significant freedom in choosing the values for b and b'.

▶ **Theorem 8.** *There is an algorithm for Problem 1 running in expected time $O(n + m)$.*

Proof. We will always take $\mu := 20 \cdot N/b^3$ and $\mu' := 20 \cdot N/(b')^3$, so we can drop the μ and μ' entries in the time complexity for each subroutine/oracle. We begin by reducing from Problem 1 to an instance of MaxRegion over a $(20 \cdot N/2^3, 2, \mathcal{T})$-block structure, which requires no work beyond computing the area prefix sums because we can take $S_{\text{Bad}} = [[\lceil N/2 \rceil]]$ and $S_{\text{Good}} = \emptyset$. Thus the total expected running time for Problem 1 will be $O(N) + T_{\text{2-Max}}(N, 2)$.

Before analyzing $T_{\text{2-Max}}(N, 2)$, we find a closed form expression for $T_{\text{1-Max}}(N, b)$ when $2 \leq b \leq N^{o(1)}$. As per Lemmas 5, 6, and 7, we have the following for any $2 \leq b, b' \leq N^{o(1)}$ such that b divides b':

$$T_{\text{1-Max}}(N, b) = O(N \cdot \frac{\log b' \log^2 b}{b}) + T_{\text{1-Max}}(N, b').$$

We can efficiently parallelize the algorithm for zero-dimensional MaxRegion given in the proof of Lemma 5. So via a standard application of multidimensional parametric search (see Theorem 4.4 in the full version of [35]) that uses zero-dimensional MaxRegion for the decision problem, $T_{\text{1-Max}}(N, b) = O(N/\log N)$ when $b \geq (\log N)^{C_1}$ for some constant C_1. If we take $b' = b^2$ and solve the recurrence, we get

$$T_{\text{1-Max}}(N, b) = O\left(N/\log N + \sum_{a=\log b, \ldots, O(\log \log N)} N \cdot \frac{\log^3(2^a)}{2^a}\right) = O(N \cdot \frac{\log^3 b}{b} + N/\log N).$$

Using the above along with Lemmas 5, 6, and 7, we have the following for any $2 \leq b, b' \leq N^{o(1)}$ such that b divides b':

$$T_{\text{2-Max}}(N, b) = O(N \cdot \frac{\log b' \log^3 b}{b} + N \cdot \frac{\log b'}{\log N}) + T_2(N, b').$$

If we again take $b' = b^2$ and stop to apply multidimensional parametric search when $b' = (\log N)^{C_2}$ for some constant C_2, we have $O(\log \log N)$ total recursions, and $\log b'$ never exceeds $O(\log \log N)$. Solving this in the same way as for $T_{\text{1-Max}}(N, b)$ gives:

$$T_{\text{2-Max}}(N, b) = O(N \cdot \frac{\log^4 b}{b} + N \cdot \frac{\log \log^2 N}{\log N}).$$

Clearly $T_2(N, 2) = O(N)$, and the theorem follows. ◀

5   Constructing the Point Oracle

In this section, we prove the Point Oracle Lemma:

Lemma 5 (Point Oracle, Restated). *For all $2 \leq b \leq N$, for all μ, we have:*

$$T_{0\text{-}MAX}(N, b, \mu) = O(N \cdot \frac{\log^2 b}{b} + \mu b).$$

In other words, we need to use a given $(\mu, b, \mathcal{T})$-block structure (where $\mathcal{T}$ is just a single point t^*) to calculate $\text{Area}((P + t^*) \cap Q)$. We can decompose the area function as

$$\text{Area}((P + t^*) \cap Q) = \sum_i \text{Area}((P_i + t^*) \cap Q_i) + \sum_{i \neq j} \text{Area}((P_i + t^*) \cap Q_j)$$

and then solve for each individual summation rapidly. The first summation is actually quite easy to compute by simply reading through the lists S_{Good} and S_{Bad} that are provided.

▶ **Lemma 9.** *There is an algorithm running in time $O(N/b + \mu b)$ such that, given a $(\mu, b, \mathcal{T})$-block structure, where $2 \leq b \leq N^{o(1)}$ and $\mathcal{T}$ consists of just a single point t^*, it will compute*

$$\sum_i \text{Area}((P_i + t^*) \cap Q_i),$$

where $P_1, \ldots P_{\lceil N/b \rceil}$ and $Q_1, \ldots Q_{\lceil N/b \rceil}$ are the blocks of P and Q determined by parameter b.

Proof. By definition of the block structure, we have a partition of $S = [N/b]$ into two subsets S_{Good} and S_{Bad} such that $|S_{\text{Bad}}| \leq \mu$, and for each $i \in S_{\text{Good}}$ we have a constant complexity quadratic function $f_i(t) = \text{Area}((P_i + t) \cap Q_i)$ for all $t \in \mathcal{T}$.

We can thus compute

$$\sum_{i \in S_{\text{Good}}} \text{Area}((P_i + t^*) \cap Q_i)$$

in time $O(N/b)$ by simply evaluating each $f_i(t^*)$. Since $|S_{\text{Bad}}| \leq \mu$, we can compute

$$\sum_{i \in S_{\text{Bad}}} \text{Area}((P_i + t^*) \cap Q_i)$$

by evaluating the area for each $(P_i + t^*) \cap Q_i$ directly. The time required is $O(\mu b)$ using known algorithms for intersecting two convex polygons whose edges are in sorted order. ◀

We now turn our attention to the summation $\sum_{i \neq j} \text{Area}((P_i + t^*) \cap Q_j)$. Because every term amounts to computing the intersection for two blocks with *different* angle ranges, we could use binary searches to evaluate each term in $O(\text{polylog}(b))$ time. However, this is too slow for our purposes, as there are a near quadratic number of terms. Instead, we take advantage of additional structure to decompose the summation into $O(N/b)$ intersections of *unions of blocks*, such that each intersection can still be computed via binary search.

As a tool, we need a way to efficiently intersect convex chains that meet certain conditions.

▶ **Lemma 10.** *Let A and B be any two convex polygons, and let X and Y be any two polygonal chains whose edges are a subset of the edges of A and B, respectively. If $\Lambda_A(X) \cap \Lambda_B(Y) = \emptyset$, then X and Y intersect at most twice, and we can find the intersection point(s) in time $O(\log |X| \log |Y|)$.*

Proof. Because the angle ranges for X and Y are disjoint, they will act as pseudo-disks, and hence they can intersect at most twice.

We now find the intersection point(s), if they exist. We can locate a vertex of X with respect to the convex chain Y in time $O(\log|Y|)$ via binary search. Using this as a subroutine, we then binary search on the vertices of X to find the two pairs of consecutive vertices on X that define X's intersections with Y. After applying a symmetric process to find two pairs of consecutive vertices on Y, we are done. The total time is $O(\log|X|\log|Y|)$.   ◄

Now we can show how to compute $\sum_{i\neq j}\mathrm{Area}((P_i+t^*)\cap Q_j)$.

▶ **Lemma 11.** *There is an algorithm running in time $O(N\cdot\frac{\log^2 b}{b})$ such that, given a $(\mu,b,\mathcal{T})$-block structure, where $\mathcal{T}$ consists of just a single point t^*, it will compute*

$$\sum_{i\neq j}\mathrm{Area}((P_i+t^*)\cap Q_j),$$

where $P_1,\ldots P_{\lceil N/b\rceil}$ and $Q_1,\ldots Q_{\lceil N/b\rceil}$ are the blocks of P and Q determined by parameter b.

Proof. We give a recursive algorithm for the problem. As a subroutine, we consider the problem of computing, given two integers $x,y\in[N/b]$ with $x>y$,

$$\sum_{(i,j)\in W(x,y)}\mathrm{Area}((P_i+t)\cap Q_j),\text{ where}$$

$$W(x,y):=\{(i,j)\in[N/b]\times[N/b]:i\neq j\text{ and }x\leq i,j\leq y\}.$$

Let $T(y-x,b)$ be the time complexity of this subroutine. The original summation $\sum_{i\neq j}\mathrm{Area}((P_i+t^*)\cap Q_j)$ is equivalent to computing $\sum_{(i,j)\in W(x,y)}\mathrm{Area}((P_i+t)\cap Q_j)$, so the lemma amounts to showing that $T(\lceil N/b\rceil-1,b)=O(N\cdot\frac{\log^2 b}{b})$, or more generally, $T(z,b)=O(z\log^2 b)$.

Consider an instance of this subroutine with any parameters $y>x$. Let $m:=\lceil\frac{x+y}{2}\rceil$ and $z:=y-x$. We can decompose the target summation as follows:

$$\sum_{(i,j)\in W(x,y)}\mathrm{Area}((P_i+t^*)\cap Q_j)=$$

$$\sum_{(i,j)\in W(x,m-1)}\mathrm{Area}((P_i+t^*)\cap Q_j)+\sum_{(i,j)\in W(m,y)}\mathrm{Area}((P_i+t^*)\cap Q_j)+$$

$$\sum_{x\leq i<m\leq j\leq y}\mathrm{Area}((P_i+t^*)\cap Q_j)+\sum_{x\leq j<m\leq i\leq y}\mathrm{Area}((P_i+t^*)\cap Q_j)$$

The first two summations can be evaluated in time at most $2T(\lceil\frac{y-x}{2}\rceil,b)$ via recursion (unless $x-y=1$, in which case they are automatically zero). The last two summations can actually be evaluated directly. We consider the case of the very last summation; the remaining one can be evaluated via a symmetric process. First note that

$$\sum_{x\leq j<m\leq i\leq y}\mathrm{Area}((P_i+t^*)\cap Q_j)=\mathrm{Area}(((\bigcup_{m\leq i\leq y}P_i)+t^*)\cap(\bigcup_{x\leq j<m}Q_j)).$$

Let $A=\bigcup_{x\leq i<m}P_i$ and $B=\bigcup_{m\leq j\leq y}Q_j$. Observe that A and B are both convex polygons, since each is the union of consecutive blocks. By construction, the edges that A shares with P and the edges that B shares with Q form two convex polygonal chains that have disjoint angle ranges, i.e. $\Lambda_P(E_P(A))\cap\Lambda_Q(E_Q(B))=\emptyset$. These chains account for all but two edges of A and two edges of B. Thus we can decompose A and B into $O(1)$

convex polygonal chains satisfying the preconditions of Lemma 10. This imples that A and B intersect in at most a constant number of locations, and furthermore we can find the intersection points in time $O(\log^2 z)$.

Using the area prefix sums provided in the block structure and known techniques for intersecting convex polygons, we can find the area in $O(1)$ additional time (the area prefix sums for P and Q also act as area prefix sums for A and B). So the recurrence is $T(z,b) = 2T(\lceil z/2 \rceil, b) + O(\log^2 z)$ when $z > 1$ and $T(1,b) = O(\log^2 b)$. This solves to $T(z,b) = O(z \log^2 b)$.   ◀

6   Constructing the Cascading Subroutine

The last step is to give an algorithm for the cascading subroutine.

Lemma 6 (Cascading Subroutine, Restated). *For all $2 \le b, b' \le N$ such that b divides b', for all μ, we have:*

$$T_{CASCADE}(N, b, b', \mu) = O(N \cdot \frac{\log b' \log b}{b} + \mu b^2).$$

In other words, given a $(\mu, b, \mathcal{T})$-block structure, parameters μ' and b', and a triangle or line segment $\mathcal{T}'$, we want to either produce a valid $(\mu', b', \mathcal{T}')$-block structure, or report that more than $\mu'/2$ lines of the set $\mathcal{S}$ intersect $\mathcal{T}'$. Throughout, let $P_1, \ldots P_{\lceil N/b \rceil}$ and $Q_1, \ldots Q_{\lceil N/b \rceil}$ be the blocks of P and Q determined by block size parameter b, and let $P'_1, \ldots P'_{\lceil N/b' \rceil}$ and $Q'_1, \ldots Q'_{\lceil N/b' \rceil}$ be the blocks of P and Q determined by block size parameter b'.

First, we show that constant complexity quadratic functions may actually be used to summarize the area function for certain block pairs.

▶ **Lemma 12.** *Let A and B be any convex polygons, and let $\mathcal{T}'$ be any triangle or line segment contained in one cell of $\Pi(A, B)$.[8] Then there exists a constant complexity quadratic function f such that $f(t) = \mathrm{Area}((A + t) \cap B)$ for all $t \in \mathcal{T}'$.*

Proof. By definition of $\Pi(A, B)$ (see Equation 1), if $\mathcal{T}'$ is contained within a single cell, then every translation $t \in \mathcal{T}'$ has the same pairs of edges from A and B intersecting when $A + t$ is intersected with B. The gradient for the function $\mathrm{Area}((A + t) \cap B)$, when t is restricted to $\mathcal{T}'$, is precisely the sum of gradients resulting from each of these edge pairs. Each individual gradient is a linear function in t, so the overall gradient is as well. Integrating this gives a function $f(t) = \mathrm{Area}((A + t) \cap B)$ that is quadratic in t.   ◀

Before describing how to perform cascading, we need a tool to help classify the new blocks based on their relationship with $\mathcal{T}'$.

▶ **Lemma 13.** *Let A and B be any two convex polygons, and let X and Y be any two polygonal chains whose edges are a subset of the edges of A and B, respectively. Let $\mathcal{T}'$ be any triangle or line segment. If $\Lambda_A(X) \cap \Lambda_B(Y) = \emptyset$, then we can determine if some line segment in $\Pi(X, Y)$ intersects the interior of $\mathcal{T}'$ in time $O(\log |X| \log |Y|)$.*

Proof. Let the set of vertices for $\mathcal{T}'$ be $V(\mathcal{T}')$. For each $t \in V(\mathcal{T}')$, we know that, by Lemma 10, $X + t$ and Y can intersect in at most two points, and we can find the two pairs of edges that contain these intersection(s) in time $O(\log |X| \log |Y|)$.[9] Let the set of all $O(1)$

---

[8] It is permissible for $\mathcal{T}'$ to touch the boundary of one cell, as long as it doesn't strictly intersect.
[9] There is an edge case where at least one of the intersections occurs at a vertex of X or a vertex of Y, but this can be handled with slightly more work.

edge pairs (e_X, e_Y) for all vertices in $V(\mathcal{T}')$ be L. We can determine if $\mathcal{T}' \subseteq \pi(e_X, e_Y)$ for all $(e_X, e_Y) \in L$ via brute force in time $O(1)$. If this is *not* the case, then by definition some line segment of $\Pi(X, Y)$ intersects the interior of $\mathcal{T}'$, and we are done. If this *is* the case, then all translations $t \in \mathcal{T}'$ realize the same intersecting pairs as the translations $t \in V(\mathcal{T}')$.

It still remains to determine if any translations $t \in \mathcal{T}'$ realize an additional intersecting pair that was not accounted for in the above. Let X^- be X with its edges participating in L deleted, and let Y^- be Y with its edges participating in L deleted. In time $O(\log |X| \log |Y|)$ we can use techniques similar to the ones given above to determine if the interior of the (implicitly generated) Minkowski sum of $\mathcal{T}'$ and X^- has any intersection with Y, and then if the interior of the (implicitly generated) Minkowski sum of $\mathcal{T}'$ and X has any intersection with Y^-. If an intersection does exist, then some line of $\Pi(X, Y)$ intersects the interior of $\mathcal{T}'$; otherwise, no line of $\Pi(X, Y)$ intersects the interior of $\mathcal{T}'$.   ◄

We are now ready to prove Lemma 6. The main observation is that the new quadratic functions stored in this data structure only need to apply when $t \in \mathcal{T}'$, not the more general case that $t \in \mathcal{T}$, so many of the previously difficult blocks can now be handled. We will find a quadratic function for almost all indices $k \in [N/b']$ in two steps:

1. Scan through the information provided in the input $(\mu, b, \mathcal{T})$ block structure, and use this to recover part of the function $\mathrm{Area}((P_k' + t) \cap Q_k')$.
2. Recover the missing information for $\mathrm{Area}((P_k' + t) \cap Q_k')$ by decomposing $\mathrm{Area}((P_k' + t) \cap Q_k')$ into the intersection of $o(b')$ convex polygons that have disjoint angle ranges. We evaluate these using layered binary searches and area prefix sums.

Proof. For all $k \in [N/b']$, note that because b divides b', we have a set of indices $C_k = \{(k-1) \cdot \frac{b'}{b} + 1, \ldots, k \cdot \frac{b'}{b}\}$ such that $P_k' = \bigcup_{i \in C_k} P_i$ and $Q_k' = \bigcup_{j \in C_k} Q_j$.[10] If for all $(i, j) \in C_k \times C_k$ for all and $t \in \mathcal{T}'$ we can summarize $\mathrm{Area}((P_i + t) \cap Q_j)$ with a constant complexity quadratic function, then we can simply add these to get a new function $f_k(t) = \mathrm{Area}((P_k' + t) \cap Q_k')$.

We first argue that each quadratic function $f_k(t)$ can be found efficiently, if it exists. Consider just a single value of k. We use the decomposition

$$f_k(t) = \sum_{(i,j) \in C_k \times C_k} \mathrm{Area}((P_i + t) \cap Q_j) = \sum_{i \in C_k} \mathrm{Area}((P_i + t) \cap Q_i) + \sum_{(i,j) \in C_k \times C_k : i \neq j} \mathrm{Area}((P_i + t) \cap Q_j)$$

For the first summation on the right hand side, we scan through the list S_{Good} from the original $(\mu, b, \mathcal{T})$-block structure. Each function $\mathrm{Area}((P_i + t) \cap Q_i)$ with $i \in S_{\mathrm{Good}}$ already has a quadratic function, and reading these takes time $O(\frac{b'}{b})$ for each k, or $O(N/b)$ time for all k. Globally, there are at most μ indices i not in S_{Good}. For each of these, we can check whether $\mathcal{T}'$ lies in a single cell of $\Pi(P_i, Q_i)$ via directly examining each line segment in $\Pi(P_i, Q_i)$ in time $O(b^2)$. If this is the case, then by Lemma 12, $\mathrm{Area}((P_i + t) \cap Q_i)$ can be summarized by a quadratic function for all $t \in \mathcal{T}'$, and we find the function via directly examining each parallelogram in $\Pi(P_i, Q_i)$ in time $O(b^2)$. The total additional time is $O(\mu b^2)$.

Now consider the second summation on the right hand side. We can rewrite this as

$$\sum_{(i,j) \in C_k \times C_k : i \neq j} \mathrm{Area}((P_i + t) \cap Q_j) = \sum_{i \in C_k} \left(\mathrm{Area}((P_i + t) \cap Q_{i,k}^-) + \mathrm{Area}((P_i + t) \cap Q_{i,k}^+) \right),$$

where $Q_{i,k}^-$ is the union of all Q_j's in the original sum with $j < i$, and $Q_{i,k}^+$ is the union of all Q_j's in the original sum with $j > i$. We evaluate each of the terms individually.

---

[10] The set may be smaller for the very last value of k, but this does not impact the proof.

Consider the case of $\text{Area}((P_i+t)\cap Q_{i,k}^-)$, as the case of $\text{Area}((P_i+t)\cap Q_{i,k}^+)$ is symmetric. $Q_{i,k}^-$ is a convex polygon, because it is the union of consecutive blocks. By construction, we have that $\Lambda_P(E_P(P_i)) \cap \Lambda_Q(E_Q(Q_{i,k}^-)) = \emptyset$. Therefore, we can break $P_i + t$ and $Q_{i,k}^-$ into $O(1)$ pieces that satisfy the preconditions of Lemma 10, showing that $P_i + t$ and $Q_{i,k}^-$ can only intersect at a constant number of locations for any translation $t \in \mathbb{R}^2$. These same pieces allow us to apply Lemma 13 to determine if $\mathcal{T}'$ lies in a single cell of $\Pi(P_i, Q_{i,k}^-)$, which takes time $O(\log|P_i|\log|Q_{i,k}^-|) = O(\log b \log b')$. If this is the case, then by Lemma 12 there exists a quadratic function equal to $\text{Area}((P_i + t) \cap Q_{i,k}^-)$ for all $t \in \mathcal{T}'$. The nonconstant components of this quadratic function only depend on the $O(1)$ intersecting pairs of edges, and we already found these via Lemma 13. Thus we can use known decomposition techniques for convex polygons, along with the area prefix sums from the $(\mu, b, \mathcal{T})$-block structure (which also serve as area prefix sums for P_i and $Q_{i,k}^-$), to find the quadratic function equal to $\text{Area}((P_i + t) \cap Q_{i,k}^-)$ in $O(1)$ additional time.

Adding the time across all terms for all $k \in [N/b']$ gives an overall time of $O(N \cdot \frac{\log b \log b'}{b})$.

Every index k for which we found a function $f_k(t)$ will be added to S'_{Good}, and we store its function. The remaining indices will be added to S'_{Bad}. The last step is to argue that $|S'_{\text{Bad}}| \leq \mu'$, meaning we have constructed a valid $(\mu', b', \mathcal{T}')$-block structure.

By construction, every $k \in S'_{\text{Bad}}$ corresponds to an intersection between $\bigcup_{(i,j)\in C_k \times C_k} \Pi(P_i, Q_j)$ and the interior of $\mathcal{T}'$. By construction (see Equation 2), each line $\overleftrightarrow{\mathcal{L}}$ in $\mathcal{S}$ has at most two indices k such $\overleftrightarrow{\mathcal{L}}$ is the line extension of some line segment in $\bigcup_{(i,j)\in C_k \times C_k} \Pi(P_i, Q_j)$. Therefore we can charge each $k \in S'_{\text{Bad}}$ to different lines of $\mathcal{S}$ in such a way that each line of $\mathcal{S}$ will be charged at most twice. Thus if $|S'_{\text{Bad}}| > \mu'$, this implies that more than $\mu'/2$ lines of $\mathcal{S}$ intersect the interior of $\mathcal{T}'$, so the algorithm can just return failure in this case. ◀

References

1   Pankaj K. Agarwal, Torben Hagerup, Rahul Ray, Micha Sharir, Michiel H. M. Smid, and Emo Welzl. Translating a planar object to maximize point containment. In Rolf H. Möhring and Rajeev Raman, editors, *Proc. 10th Annual European Symposium on Algorithms (ESA)*, volume 2461 of *Lecture Notes in Computer Science*, pages 42–53. Springer, 2002. `doi:10.1007/3-540-45749-6\_8`.

2   Pankaj K. Agarwal and Micha Sharir. Efficient algorithms for geometric optimization. *ACM Comput. Surv.*, 30(4):412–458, 1998. `doi:10.1145/299917.299918`.

3   Hee-Kap Ahn, Peter Brass, and Chan-Su Shin. Maximum overlap and minimum convex hull of two convex polyhedra under translations. *Comput. Geom.*, 40(2):171–177, 2008. `doi:10.1016/J.COMGEO.2007.08.001`.

4   Hee-Kap Ahn, Siu-Wing Cheng, Hyuk Jun Kweon, and Juyoung Yon. Overlap of convex polytopes under rigid motion. *Comput. Geom.*, 47(1):15–24, 2014. `doi:10.1016/J.COMGEO.2013.08.001`.

5   Hee-Kap Ahn, Siu-Wing Cheng, and Iris Reinbacher. Maximum overlap of convex polytopes under translation. *Comput. Geom.*, 46(5):552–565, 2013. `doi:10.1016/J.COMGEO.2011.11.003`.

6   Hee-Kap Ahn, Otfried Cheong, Chong-Dae Park, Chan-Su Shin, and Antoine Vigneron. Maximizing the overlap of two planar convex sets under rigid motions. *Comput. Geom.*, 37(1):3–15, 2007. `doi:10.1016/J.COMGEO.2006.01.005`.

7   Helmut Alt and Leonidas J. Guibas. Discrete geometric shapes: Matching, interpolation, and approximation. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 121–153. North Holland / Elsevier, 2000. `doi:10.1016/B978-044482537-7/50004-8`.

8 Nancy M. Amato and Franco P. Preparata. A time-optimal parallel algorithm for three-dimensional convex hulls. *Algorithmica*, 14(2):169–182, 1995. `doi:10.1007/BF01293667`.

9 Gill Barequet, Matthew T. Dickerson, and Petru Pau. Translating a convex polygon to contain a maximum number of points. *Comput. Geom.*, 8:167–179, 1997. `doi:10.1016/S0925-7721(96)00011-9`.

10 Sergio Cabello, Mark de Berg, Panos Giannopoulos, Christian Knauer, René van Oostrum, and Remco C. Veltkamp. Maximizing the area of overlap of two unions of disks under rigid motion. *Int. J. Comput. Geom. Appl.*, 19(6):533–556, 2009. `doi:10.1142/S0218195909003118`.

11 Timothy M. Chan. A simpler linear-time algorithm for intersecting two convex polyhedra in three dimensions. *Discret. Comput. Geom.*, 56(4):860–865, 2016. `doi:10.1007/S00454-016-9785-3`.

12 Timothy M. Chan. (Near-)linear-time randomized algorithms for row minima in Monge partial matrices and related problems. In *Proc. of the 32nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1465–1482, 2021. `doi:10.1137/1.9781611976465.88`.

13 Timothy M. Chan and Isaac M. Hair. Convex polygon containment: Improving quadratic to near linear time. In *Proc. 40th International Symposium on Computational Geometry (SoCG)*, volume 293 of *LIPIcs*, pages 34:1–34:15, 2024. `doi:10.4230/LIPICS.SOCG.2024.34`.

14 Bernard Chazelle. The polygon containment problem. *Advances in Computing Research*, 1(1):1–33, 1983. URL: `https://www.cs.princeton.edu/~chazelle/pubs/PolygContainmentProb.pdf`.

15 Bernard Chazelle. Triangulating a simple polygon in linear time. *Discret. Comput. Geom.*, 6:485–524, 1991. `doi:10.1007/BF02574703`.

16 Bernard Chazelle. Cuttings. In Dinesh P. Mehta and Sartaj Sahni, editors, *Handbook of Data Structures and Applications*. Chapman and Hall/CRC, 2004. `doi:10.1201/9781420035179.CH25`.

17 Bernard Chazelle and Leonidas J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1(2):133–162, 1986. `doi:10.1007/BF01840440`.

18 Siu-Wing Cheng and Chi-Kit Lam. Shape matching under rigid motion. *Comput. Geom.*, 46(6):591–603, 2013. `doi:10.1016/J.COMGEO.2013.01.002`.

19 Kenneth L. Clarkson and Peter W. Shor. Application of random sampling in computational geometry, II. *Discret. Comput. Geom.*, 4:387–421, 1989. `doi:10.1007/BF02187740`.

20 Mark de Berg, Otfried Cheong, Olivier Devillers, Marc J. van Kreveld, and Monique Teillaud. Computing the maximum overlap of two convex polygons under translations. *Theory Comput. Syst.*, 31(5):613–628, 1998. `doi:10.1007/PL00005845`.

21 Martin E. Dyer. Linear time algorithms for two- and three-variable linear programs. *SIAM J. Comput.*, 13(1):31–45, 1984. `doi:10.1137/0213003`.

22 Martin E. Dyer. On a multidimensional search technique and its application to the euclidean one-centre problem. *SIAM J. Comput.*, 15(3):725–738, 1986. `doi:10.1137/0215052`.

23 Greg N. Frederickson and Donald B. Johnson. Generalized selection and ranking: Sorted matrices. *SIAM J. Comput.*, 13(1):14–30, 1984. `doi:10.1137/0213002`.

24 Sariel Har-Peled and Subhro Roy. Approximating the maximum overlap of polygons under translation. *Algorithmica*, 78(1):147–165, 2017. `doi:10.1007/S00453-016-0152-9`.

25 Shreesh Jadhav and Asish Mukhopadhyay. Computing a centerpoint of a finite planar set of points in linear time. *Discret. Comput. Geom.*, 12:291–312, 1994. `doi:10.1007/BF02574382`.

26 Mook Kwon Jung, Seokyun Kang, and Hee-Kap Ahn. Minimum convex hull and maximum overlap of two convex polytopes. In *Proc. 36th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2025. To appear.

27 Garam Kim, Jongmin Choi, and Hee-Kap Ahn. Maximum overlap of two convex polygons. *KIISE Transactions on Computing Practices*, 27(8):400–405, 2021. `doi:10.5626/KTCP.2021.27.8.400`.

28 Nimrod Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30(4):852–865, 1983. `doi:10.1145/2157.322410`.

29   Nimrod Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM J. Comput.*, 12(4):759–776, 1983. `doi:10.1137/0212052`.

30   Nimrod Megiddo. Linear programming in linear time when the dimension is fixed. *J. ACM*, 31(1):114–127, 1984. `doi:10.1145/2422.322418`.

31   Nimrod Megiddo. Partitioning with two lines in the plane. *J. Algorithms*, 6(3):430–433, 1985. `doi:10.1016/0196-6774(85)90011-2`.

32   David M. Mount, Ruth Silverman, and Angela Y. Wu. On the area of overlap of translated polygons. *Comput. Vis. Image Underst.*, 64(1):53–61, 1996. `doi:10.1006/CVIU.1996.0045`.

33   Wlodzimierz Ogryczak and Arie Tamir. Minimizing the sum of the k largest functions in linear time. *Inf. Process. Lett.*, 85(3):117–122, 2003. `doi:10.1016/S0020-0190(02)00370-8`.

34   Joseph O'Rourke, Subhash Suri, and Csaba D. Tóth. Polygons. In Jacob E. Goodman, Joseph O'Rourke, and Csaba D. Tóth, editors, *Handbook of Discrete and Computational Geometry*, pages 787–810. Chapman and Hall/CRC, 3rd edition edition, 2017. URL: `https://www.csun.edu/~ctoth/Handbook/chap30.pdf`.

35   Sivan Toledo. Maximizing non-linear concave functions in fixed dimension. In *Proc. 33rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 676–685, 1992. `doi:10.1109/SFCS.1992.267783`.

36   Eitan Zemel. An $O(n)$ algorithm for the linear multiple choice knapsack problem and related problems. *Inf. Process. Lett.*, 18(3):123–128, 1984. `doi:10.1016/0020-0190(84)90014-0`.

37   Honglin Zhu and Hyuk Jun Kweon. Maximum overlap area of a convex polyhedron and a convex polygon under translation. In *Proc. 39th International Symposium on Computational Geometry (SoCG)*, volume 258 of *LIPIcs*, pages 61:1–61:16, 2023. `doi:10.4230/LIPICS.SOCG.2023.61`.