Touring a Sequence of Orthogonal Polygons

Katrin Casel
Humboldt-University Berlin, Germany

Sándor Kisfaludi-Bak
Aalto University, Espoo, Finland

Linda Kleist
University of Hamburg, Germany

Jeroen S.K. Lamme
Eindhoven University of Technology, The Netherlands

Eunjin Oh
POSTECH, Pohang, South Korea

Yanheng Wang
ETH Zurich, Switzerland

Abstract

We study the problem of computing a shortest tour that visits a sequence of k polygons $P_1, \dots, P_k$ with a total number of n vertices. A tour is an oriented curve such that there exist points $p_i \in P_i$ for all i where p_i appears not after p_{i+1}. In a seminal paper, Dror, Efrat, Lubiw and Mitchell (STOC 2003) considered the problem under L_2 distance, and gave $\widetilde{O}(nk)$ and $\widetilde{O}(nk^2)$ algorithms for disjoint and intersecting convex polygons, respectively. In this paper, we consider the orthogonal setting (with orthogonal polygons and Manhattan distance) and obtain the following results:

- a truly subquadratic $\widetilde{O}(n^{2-\frac{1}{48}})$ algorithm when consecutive polygons in the sequence are disjoint;
- an $\widetilde{O}(n)$ algorithm for ortho-convex polygons when consecutive polygons are disjoint;
- an $O(n)$ algorithm for axis-aligned rectangles;
- $\widetilde{O}(n^2)$ and $\widetilde{O}(n^{1.5}k^2)$ algorithms without restrictions.

Our algorithms build on a wide range of techniques, including additively weighted Voronoi diagrams, rectangle decompositions, persistent data structures, and dynamic distance oracles for weighted planar graphs.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Dynamic programming

Keywords and phrases shortest path, subquadratic time, dynamic planar distance oracle

Digital Object Identifier 10.4230/LIPIcs.ICALP.2026.50

Category Track A: Algorithms, Complexity and Games

Related Version *The full version is available at*: `https://arxiv.org/abs/2605.07882`

Funding *Sándor Kisfaludi-Bak*: Supported by the Research Council of Finland, Grant 363444.
Eunjin Oh: Supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.RS-2024-00440239, Sublinear Scalable Algorithms for Large-Scale Data Analysis) and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No.RS-2024-00358505).
Yanheng Wang: Part of this work was finished at Saarland University, Saarbrücken, Germany. It was part of the project TIPEA that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 850979).

Acknowledgements This work was initiated at the 2025 Lorentz Center workshop "Fine-Grained and Parameterized Computational Geometry" in Leiden, the Netherlands.

1 Introduction

Shortest paths between two points in geometric environments can often be computed in near-linear time. However, in settings where the path is constrained to avoid or pass through given regions, polynomial-time algorithms might not exist. For example, computing a shortest path that avoids axis-aligned half-planes in $\mathbb{R}^3$, or one that passes through non-convex regions in $\mathbb{R}^2$ are both NP-hard problems [30, 20].

In this paper we study a constrained shortest path problem POLYGON TOURING: given a sequence $P_1, \dots, P_k \subset \mathbb{R}^2$ of polygons with n vertices in total, compute (the length of) a shortest tour that visits the polygons in order. A *tour* π is an oriented open curve, and it *visits* a sequence of polygons $P_1, \dots, P_k$ if there exist points $p_i \in P_i$ for all $i \in \{1, \dots, k\}$ such that π goes through $p_1, \dots, p_k$ in order. We highlight that the points may coincide; see Figure 1 for an example. One can measure the length of a tour in different metrics, and natural choices include the Manhattan (L_1) and Euclidean (L_2) metrics.

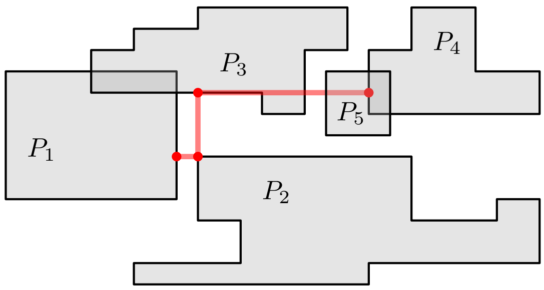

Figure 1 A tour visiting a sequence of five orthogonal polygons. All polygons but P_2 are ortho-convex. Note that deleting P_5 yields a step-disjoint instance.

POLYGON TOURING has obvious applications in motion planning and logistics in physical environments [21, 25, 27, 32]. It is also used as a subroutine in several geometric optimization problems, such as the watchman route problem [29, 31], the safari and zookeeper problems [35, 5, 22], and even variants of convex hull [28, 36, 19, 1]. The problem can also be understood as the 2-dimensional offline version of the convex body chasing problem [23, 33, 3, 2, 13, 34].

Dror, Efrat, Lubiw and Mitchell [20] studied POLYGON TOURING under L_2 when the polygons are convex. In their formulation the starting point of the tour is fixed, i.e., P_1 degenerates to a single point. They gave an $O(nk \log n)$-time algorithm if the given convex polygons are disjoint, and an $O(nk^2 \log n)$-time algorithm if they are allowed to overlap. On the other hand, they showed that the problem becomes NP-hard if the polygons are non-convex, or even if each P_i consists of a pair of segments with a shared endpoint.

This raises interesting questions from a fine-grained perspective. For example, is there a *truly subquadratic time* algorithm (i.e., of running time $O(n^{2-\varepsilon})$ for some $\varepsilon > 0$) that solves POLYGON TOURING under L_2, assuming convex disjoint polygons of constant size each?

As our take to the question, we consider an interesting variant of touring *orthogonal polygons* (where every edge is either horizontal or vertical) under L_1. This setting avoids precision issues under L_2 and allows us to concentrate on the combinatorial side of the problem. Here obtaining an $\widetilde{O}(n^2)$-time algorithm is in fact an open problem. Dror *et al.* [20] claimed an $O(n^2)$-time algorithm but did not present a full proof. A straightforward interpretation of their sketch would lead to an algorithm in cubic time (more precisely, $O(n^2k)$ time).

▶ **Question 1.** *Is there an $\widetilde{O}(n^2)$-time algorithm for touring a sequence of orthogonal polygons under L_1? What about $\widetilde{O}(nk)$ or even truly-subquadratic time?*

Table 1 Overview of our results for Orthogonal Polygon Touring(n, k).

input restriction	time complexity	refer to
–	$O(n^2 \log^2 n)$	Theorem 1
–	$\widetilde{O}(n^{1.5}k^2)$	Theorem 4
step-disjoint	$\widetilde{O}(n^{2-\frac{1}{48}})$	Theorem 3
step-disjoint, ortho-convex	$\widetilde{O}(n)$	Theorem 5
rectangles	$O(n)$	Theorem 2

Guided by known results in L_2, we can expect that the problem becomes easier when the polygons are disjoint and/or convex. In case of orthogonal polygons the only convex polygons are axis-aligned rectangles. This leads us to the second question:

▶ **Question 2.** *Is there a faster algorithm for touring a sequence of* disjoint *orthogonal polygons under L_1? What about touring a sequence of rectangles under L_1?*

Our contribution

In what follows, we consider Polygon Touring for a sequence of k orthogonal polygons with n vertices in total and measure all distances in L_1 metric. We denote this problem by Orthogonal Polygon Touring(n, k). Table 1 summarizes our results. As a baseline, we design an $\widetilde{O}(n^2)$-time algorithm, thereby partially answering Question 1.

▶ **Theorem 1.** *Orthogonal Polygon Touring(n, k) is in time $O(n^2 \log^2 n)$.*

The proof of Theorem 1 already contains some non-trivial ideas that recur in the other results. First observe that we may restrict our attention to tours whose vertices lie on the the *grid*, i.e., the intersections of the horizontal and vertical lines through polygon vertices. Since the grid has $O(n^2)$ points, this leads to a naive dynamic program that runs in time $O(n^2k)$. To improve the running time it is helpful to consider the case of *step-disjoint* polygons, where any two consecutive polygons in the sequence are disjoint. (In Figure 1, $P_1, \dots, P_4$ is step-disjoint, while $P_1, \dots, P_5$ is not.) In this case, a shortest tour π must enter each polygon P_i at a boundary point which is also on the grid; we call these points *portals*. Note that the number of portals on P_i is at most n times the number of edges in P_i, so the total number of portals over all polygons is $O(n^2)$. It remains to design an efficient data structure that allows one to query, for each portal on P_{i+1}, the shortest tour ending there. Since such tour must pass through a portal on P_i, we can build an additively weighted Voronoi diagrams over the portals on P_i to handle these queries efficiently.

In the general case where consecutive polygons can overlap, a shortest tour may enter a polygon directly at a grid point in its interior; see for example P_5 in Figure 1. We handle the situation by "tunneling" through the overlapping polygons with the help of a segment-tree-based data structure.

We do not know how to push this approach to fully address Question 1. Nevertheless, we manage to obtain truly-subquadratic time algorithms under various natural restrictions. In the most restricted case, we consider a sequence of rectangles and answer the second half of Question 2:

▶ **Theorem 2.** *Orthogonal Polygon Touring(n, k) for rectangles can be solved in $O(n) = O(k)$ time.*

This highly specialized algorithm uses the convexity of rectangles in a strong way, and does not even generalize to polygons with $O(1)$ edges each (such as L-shapes). It is also challenging to solve the other extreme: a sequence of $O(1)$ polygons with $\Theta(n)$ edges each. As our main result, we propose ideas to solve both extremes for step-disjoint sequences, and to interpolate the extremes into a truly subquadratic algorithm. This answers the first half of Question 2.

▶ **Theorem 3** (Main theorem). *Orthogonal Polygon Touring(n, k) for step-disjoint orthogonal polygons can be solved in $\widetilde{O}(n^{2-\frac{1}{48}})$ time.*

The global structure of the algorithm is to split the sequence into batches of two types: a dense batch contains few polygons (but each polygon may have many edges), whereas a sparse batch contains polygons with few edges. It uses different strategies to process the two types and chain them together. For simplicity, we expose the ideas on two extreme cases: In the dense case $k \leqslant n^{0.1}$ is small (thus all polygons are in one dense batch); and in the sparse case $k = \Omega(n)$ and all polygons have $O(1)$ edges (thus all polygons are in one sparse batch).

In the dense case, we start by partitioning the plane into rectangles using a technique of De Berg and Van Kreveld [18]. The partition has the property that, for every polygon P_i and rectangle R, the intersection $P_i \cap R$ is either a collection of horizontal stripes or a collection of vertical stripes. As a result, the tour inside R is rather simple and can be reduced to one-dimensional problems. Our idea is to build a dynamic program on the grid points induced by rectangle boundaries. For each such grid point p (called a *hub*) and each index i we want to compute the shortest tour visiting $P_1, \ldots, P_i, p$. We need to iterate over index pairs $1 \leqslant i \leqslant j \leqslant k$ during the computation, and the number of hubs can be bounded by $O(n^{1.5})$, so this results in an algorithm with running time $\widetilde{O}(n^{1.5}k^2)$. Due to the quadratic dependence on k this algorithm is not subquadratic for larger values of k, so we need a completely different strategy for the sparse case.

In the sparse case, we face a long sequence of polygons with few edges. Importantly, if the tour is visiting a pair of consecutive polygons P_i and P_{i+1} and makes a turn (i.e., changes from horizontal to vertical or vice versa) somewhere between their visits, then the turn can only be justified by a *local* grid point, i.e., on some point that is the intersection of some horizontal and vertical lines through the vertices of P_i and P_{i+1}. Since both polygons have few edges, the local grid has very small complexity. Tours making a turn between each consecutive pair can be handled using weighted Voronoi diagrams.

Unfortunately, it is possible that a tour traverses a subsequence of consecutive polygons without making any turns, and hence without snapping to the local grid. Tracking such tours efficiently is far from trivial. In order to compute tours that stay on a given horizontal line $y = y^*$, we build an auxiliary planar graph whose nodes are arranged by layers that correspond to the polygon index $1 \leqslant i \leqslant k$, and inside each layer by x-coordinates that correspond to the x-coordinates where the polygons intersect the line $y = y^*$. The shortest horizontal tour on $y = y^*$ visiting the polygons corresponds to a shortest path in this graph that percolates from the first layer to the last layer. We cannot afford building these graphs from scratch for each relevant value of y^*, as this requires quadratic time. However, we observe that as we sweep a horizontal line bottom up, the number of updates to the planar graph is linear in total, and we can apply a multi-source distance oracle for dynamic planar graphs by Charalampopoulos and Karczmarz [16] to beat quadratic time.

Interpolating the dense and sparse cases requires further finesse. In particular, we need to deal with instances that contain both dense and sparse batches. A challenge is to pass information from one batch to the next and bridge the difference in strategies. The bridging is efficient only if the polygon delimiting the two batches has a small number of edges. Fortunately, we show a batching strategy such that all the delimiters have small complexity.

This concludes our overview of the main theorem. As a byproduct of the dense case we also obtain an algorithm for potentially overlapping polygons. The algorithm is truly-subquadratic (hence, faster than Theorem 1) whenever $k \leqslant n^{0.25-\varepsilon}$ for any $\varepsilon > 0$.

▶ **Theorem 4.** *Orthogonal Polygon Touring(n, k) can be solved in $\widetilde{O}(n^{1.5}k^2)$ time.*

In another result, we consider *ortho-convex* polygons, which are orthogonal polygons with the property that any horizontal or vertical line intersects them in a segment. If the ortho-convex polygons are step-disjoint, then a vertical edge S of P_i cannot have points of P_{i-1} both to its left and right. Hence, for each grid point $p \in S$, the shortest tour visiting $P_1, \dots, P_{i-1}, p$ can be efficiently described by the distance functions on the vertical edges of P_{i-1} that are "facing" S. The distance functions in turn are piecewise linear and have slopes -1, 0 or 1. By handling and updating these functions in a persistent data structure we are able to solve the step-disjoint ortho-convex case in near-linear time, avoiding the reliance on dynamic planar graph algorithms.

▶ **Theorem 5.** *Orthogonal Polygon Touring(n, k) can be solved in $O(n \log n)$ time for step-disjoint ortho-convex polygons.*

While this paper is about fast algorithms for Orthogonal Polygon Touring, the problem is also interesting from the lower bound/fine-grained complexity perspective. In particular, it does not seem to have the same quantifier structure as typical geometric problems studied in the fine-grained literature such as curve and shape similarity problems [6, 10, 11, 8, 12], intersection graph problems [9, 14], clustering [15], and point-line incidence problems [24, 4]. We raise the following natural question.

▶ **Question 3.** *Is there a super-linear conditional lower bound for touring a sequence of convex polygons under L_2, or for touring orthogonal polygons under L_1?*

Towards this direction we consider the generalized problem in 3-dimensional space: touring a sequence of k orthogonal polytopes of n vertices in total. The problem has a straightforward $\widetilde{O}(n^3k)$ algorithm. We show a conditional lower bound assuming the orthogonal vectors hypothesis (OVH). See [7] for an overview of popular fine-grained conjectures.

▶ **Theorem 6.** *Assuming OVH, no algorithm can solve Orthogonal Polytope Touring(n, k) for step-disjoint 3-dimensional polytopes in time $O(n^{2-\epsilon}\mathrm{poly}(k))$, for any $\varepsilon > 0$.*

Organization. Section 2 introduces the fundamental concepts and structures. Section 3 studies the general case without input restriction. Sections 4 and 5 assume step-disjointness and present truly-subquadratic time algorithms for orthogonal polygons and ortho-convex polygons, respectively. Along the way we also obtain an algorithm that handles (a small number of) overlapping polygons. The remaining results (Theorems 2 and 6) are presented in the full version of this paper. A few lemmas are marked with asterisks: their proofs are straightforward and thus omitted. The reader may find them in the full version as well.

2 Fundamental concepts

We write $[N] := \{1, \dots, N\}$. A polygon is *orthogonal* if all its edges are horizontal or vertical. For simplicity of presentation, we assume that all polygons are in general position, i.e., no two edges are collinear. The assumption can be removed by imposing an ordering on collinear edges. For a polygon P, let $x(P)$ and $y(P)$ be the sets of x and y-coordinates of the vertices of P, respectively. Denote $n(P) := |x(P)| = |y(P)|$. In the Orthogonal Polygon Touring(n, k) problem, we are given orthogonal polygons $P_1, \ldots, P_k$ with total complexity $n := \sum_{i=1}^{k} n(P_i)$, and the goal is to compute (the length of) a shortest tour under L_1 metric that visits $P_1, \ldots, P_k$. We say that the input sequence of polygons is *step-disjoint* if P_i and P_{i+1} are disjoint for all $1 \leqslant i \leqslant k-1$.

For each index $i \in [k]$ and point p, we define $f_i(p)$ as the length of a shortest tour that visits $P_1, \ldots, P_{i-1}, p$ in sequence. We also define $\mathrm{next}(i, p) := \min\{j \geqslant i : p \notin P_j\}$. (If j does not exist then let $\mathrm{next}(i, p) := k + 1$.) These definitions play a central role in our algorithms.

It is sometimes convenient to represent tours as discrete sequences instead of continuous curves. The notion of skeletons serves this purpose.

▶ **Definition 7.** *A skeleton visiting $P_1, \ldots, P_k$ is a sequence $(i_1, q_1), \ldots, (i_m, q_m)$ such that $1 = i_1 \leqslant i_2, \ldots, i_m < i_{m+1} := k + 1$ and $q_t \in \bigcap_{i_t \leqslant i < i_{t+1}} P_i$ for all $t \in [m]$. We say that the skeleton has* size *m and* length *$\sum_{i=2}^{m} \mathrm{dist}(q_{i-1}, q_i)$.*

So far as the minimum length is concerned, skeletons and tours are equivalent:

▶ **Lemma 8** (∗). *The minimum length of tours visiting $P_1, \ldots, P_k$ is equal to the minimum length of skeletons visiting $P_1, \ldots, P_k$.*

The next lemma provides crucial insights into the structure of minimum length skeletons.

▶ **Lemma 9.** *Among all minimum length skeletons visiting $P_1, \ldots, P_k$, there is a skeleton $(i_1, q_1), \ldots, (i_m, q_m)$ with the four properties listed below. Here we denote $Q_t := \bigcap_{i_t \leqslant i < i_{t+1}} P_i$.*

1. *For each $t \in [m]$, we have $\mathrm{next}(i_t, q_t) = i_{t+1}$.*
2. *For each $t \in [m]$, we have $q_t \in \partial Q_t$.*
3. *For each $t \in [2, m]$, at least one of the following holds:*
   (1) *q_t is a vertex of Q_t or an intersection point in $\partial Q_{t-1} \cap \partial Q_t$;*
   (2) *q_{t-1}, q_t are on horizontal edges of Q_{t-1}, Q_t respectively, and $x(q_{t-1}) = x(q_t)$;*
   (3) *q_{t-1}, q_t are on vertical edges of Q_{t-1}, Q_t respectively, and $y(q_{t-1}) = y(q_t)$;*
4. *There exists $t \in [m]$ for which (1) holds.*

Proof. Among all minimum length skeletons $(i_1, q_1), \ldots, (i_m, q_m)$, pick those with the minimum size m. Among all picked skeletons, further pick those such that $(i_2 - i_1, \ldots, i_{m+1} - i_m)$ is lexicographically maximum. We call these skeletons *irreducible.*

▷ Claim. Every irreducible skeleton $(i_1, q_1), \ldots, (i_m, q_m)$ satisfies property 1 and has $q_1 \in \partial Q_1$.

Proof. On one hand, $\mathrm{next}(i_t, q_t) \geqslant i_{t+1}$ because $q_t \in Q_t = \bigcap_{i_t \leqslant i < i_{t+1}} P_i$ by the definition of skeletons. On the other hand, $\mathrm{next}(i_t, q_t) \leqslant i_{t+1}$ because otherwise we can replace i_{t+1} with $\mathrm{next}(i_t, q_t)$ in the skeleton, which results in another skeleton with the same length and size but a higher lexicographic rank, a contradiction to irreducibility. This shows $\mathrm{next}(i_t, q_t) = i_{t+1}$.

Next we show $q_1 \in \partial Q_1$. Note that $q_1 \notin P_{i_2}$ since $\mathrm{next}(1, q_1) = i_2$. In particular, $q_1 \notin Q_2$. Suppose to contradiction that $q_1 \notin \partial Q_1$, then it must be in the interior of $Q_1 \backslash Q_2$. As $q_2 \in Q_2$, we can move q_1 towards q_2 while staying inside Q_1. The skeleton still visits $P_1, \ldots, P_k$ but the length strictly decreases, which is a contradiction. ◁

In the rest of the proof, we will start from an arbitrary irreducible skeleton σ and modify it into another irreducible skeleton σ' that is closer to satisfying properties 2 and 3. In more detail, we will *move* a point q_t to q'_t, meaning that we replace q_t with q'_t in σ to obtain σ'. We will make sure that $q'_t \in Q_t$ and σ' has the same length as σ. Once these two conditions are met, it follows that σ' still visits $P_1, \ldots, P_k$ and has the same length, size and lexicographic rank as σ, so it remains irreducible.

To implement the scheme, we first iterate over $t \in [m-1]$. We have $q_t \notin P_{i_{t+1}}$ because $\mathrm{next}(i_t, q_t) = i_{t+1}$ by property 1. In particular, $q_t \notin Q_{t+1}$. Since $q_{t+1} \in Q_{t+1}$, the shortest path between q_t, q_{t+1} must cross the boundary ∂Q_{t+1}. Therefore, we can move q_{t+1} to ∂Q_{t+1} while preserving the length. The resulting skeleton is still irreducible. After all iterations, we obtain an irreducible skeleton with property 2.

Next we iterate over $s = 2, \ldots, m$. In step s, the goal is to move q_{s-1}, q_s such that for every $t \in [2, s]$ either (1), (2) or (3) holds. By property 2, q_{s-1} and q_s are on edges $e_{s-1} \subset \partial Q_{s-1}$ and $e_s \subset \partial Q_s$, respectively. Assume that e_s is horizontal; the vertical case is symmetric. We distinguish four cases, illustrated in the four columns of Figure 2.

- If e_{s-1} is horizontal, then we move q_s along e_s towards q_{s-1} until it hits a vertex of Q_s, or until $x(q_{s-1}) = x(q_s)$. This ensures (1) or (2) for $t = s$.
- If e_{s-1} is vertical and entirely below/above e_s, then we move q_{s-1} to the top/bottom endpoint of e_{s-1}. Now that q_{s-1} is a vertex, (1) holds for $t = s-1$; moreover, q_{s-1} must lie on a horizontal edge, so we can apply the previous case to move q_s and ensure (1) or (2) for $t = s$.
- If e_{s-1} is vertical and entirely to the left/right of e_s, then we move q_s to the left/right endpoint of e_s and ensure (1) for $t = s$.
- If e_{s-1} is vertical and intersects e_s, then we move q_s to the intersection and ensure (1) for $t = s$.

In any case the goal is achieved. Furthermore, q_{s-1}, q_s remain on $\partial Q_{s-1}, \partial Q_s$, respectively, and the length of the skeleton does not change. Therefore, the resulting skeleton is still irreducible and satisfies property 2. After all iterations, we obtain an irreducible skeleton that satisfies both properties 2 and 3.

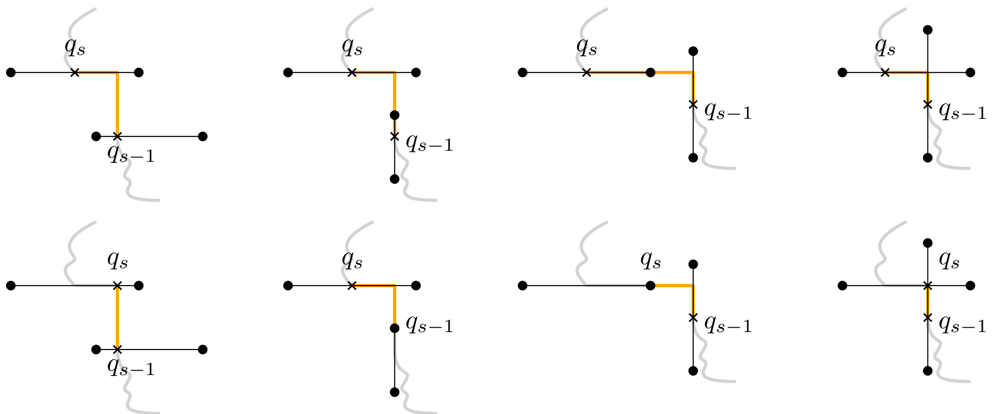

Figure 2 The four cases when e_s is horizontal. The first and second rows illustrate the situation before and after the move, respectively. The orange path in the background illustrates a shortest path between q_{s-1} and q_s.

Finally, suppose that property 4 is not yet fulfilled. Then for all $t \in [2, m]$ either (2) or (3) holds. Assume by symmetry that q_1 is on a horizontal edge of Q_1, then it is not on a vertical edge as it is not a vertex. Hence q_2 must be on a horizontal edge of Q_2 and $x(q_1) = x(q_2)$. Propagating the argument, all $q_1, \ldots, q_m$ are on horizontal edges and have the same x-coordinate. We move all points to the right by the same distance, until some point q_t becomes a vertex of Q_t, which witnesses property 4. Clearly every point q_s remains on a horizontal edge of Q_s and has the same x-coordinate, and the length of the skeleton does not change. Therefore, irreducibility and properties 2, 3 are also preserved. ◀

Motivated by Lemma 9, we define the *grid* of a set of orthogonal polygons $\mathcal{P}$ as

$$\operatorname{grid}(\mathcal{P}) := \left(\bigcup_{P\in\mathcal{P}} x(P)\right) \times \left(\bigcup_{P\in\mathcal{P}} y(P)\right) \subset \mathbb{R}^2.$$

We define the *portals* of P_i as $\operatorname{port}_i := \partial P_i \cap \operatorname{grid}(\{P_1,\ldots,P_k\})$.

▶ **Corollary 10.** *Among all minimum length skeletons visiting $P_1,\ldots,P_k$, there is a skeleton $(i_1,q_1),\ldots,(i_m,q_m)$ with the four properties in Lemma 9, as well as the property that $q_t \in \bigcup_{i_t\leqslant i<i_{t+1}} \operatorname{port}_i$ for all $t\in[m]$.*

Proof. Take a skeleton $(i_1,q_1),\ldots,(i_m,q_m)$ given by Lemma 9. By property 4, some of $q_1,\ldots,q_m$ are vertices or intersections. We split the skeleton at these points into pieces. Consider any piece but the first one. The piece starts with a point q_t that is a vertex or an intersection, and no other point is a vertex or an intersection. By property 3, points in this piece either all lie on horizontal edges and have x-coordinate $x(q_t)\in\bigcup_{i=1}^k x(P_i)$, or all lie on vertical edges and have y-coordinate $y(q_t)\in\bigcup_{i=1}^k y(P_i)$. Therefore, every point q_s in the piece is a portal. Moreover, since $q_s\in\partial Q_s\subseteq\bigcup_{i_s\leqslant i<i_{s+1}}\partial P_i$, we have $q_s\in\bigcup_{i_s\leqslant i<i_{s+1}}\operatorname{port}_i$. For the first piece, we can make the same argument from where it ends. ◀

When the polygon sequence is step-disjoint, we get a significantly simpler structure:

▶ **Corollary 11.** *Among all shortest tours that visit a sequence of step-disjoint polygons $P_1,\ldots,P_k$, there is a tour that visits a sequence of points $p_1\in\operatorname{port}_1,\ldots,p_k\in\operatorname{port}_k$ such that for each $i\in[2,k]$,*

(1) *p_i is a vertex of P_i; or*

(2) *p_{i-1},p_i are on horizontal edges of P_{i-1},P_i respectively, and $x(p_{i-1})=x(p_i)$; or*

(3) *p_{i-1},p_i are on vertical edges of P_{i-1},P_i respectively, and $y(p_{i-1})=y(p_i)$.*

Proof. Take a skeleton $(i_1,p_1),\ldots,(i_m,p_m)$ given by Corollary 10. Since the polygon sequence is step-disjoint, we have $\operatorname{next}(i_t,p_t)=i_t+1$ for all t by definition, so inductively $i_t=t$. Hence the skeleton has form $(1,p_1),(2,p_2),\ldots,(k,p_k)$, and for all $i\in[k]$ we have $Q_i=P_i$ and $p_i\in\operatorname{port}_i$. ◀

3 Touring overlapping orthogonal polygons

In this section, we study Orthogonal Polygon Touring for the general case. Dror *et al.* [20] stated that the problem can be solved in $O(n^2)$ time without elaborating on the proof; a straightforward interpretation of their idea actually needs $O(n^2k)$ time. We describe here a simple algorithm in time $O(n^2\log^2 n)$, which essentially closes the gap and also serves as a baseline for our other algorithms.

▶ **Theorem 1.** *Orthogonal Polygon Touring(n,k) is in time $O(n^2\log^2 n)$.*

Proof. As a preparation, we construct a binary tree, where each leaf node represents an element (i.e., a singleton interval) of $[k]$, and each non-leaf node represents the union of the intervals of its two children; this is a discrete variant of a segment tree [17]. We write $I(u)$ for the interval represented by node u and define $Q(u) := \bigcap_{i\in I(u)} P_i$. Note that $Q(u)=Q(v)\cap Q(w)$ where v,w are the children of u, so we can compute $Q(u)$ for all $u\in T$ in a bottom-up fashion. Then we build a point location data structure for each $Q(u)$.

It is well-known that the intersection of two orthogonal polygons P and Q can be computed in time $O\left((n(P)+n(Q))^2\right)$, and a point-location data structure on a polygon P can be built in time $O(n(P)\log n(P))$. So the total time spent per level of T is $O\left(\left(\sum_{i=1}^{k} n(P_i)\right)^2 \log n\right) = O(n^2 \log n)$. As there are $\lceil \log k \rceil$ levels, the construction time is $O(n^2 \log^2 n)$.

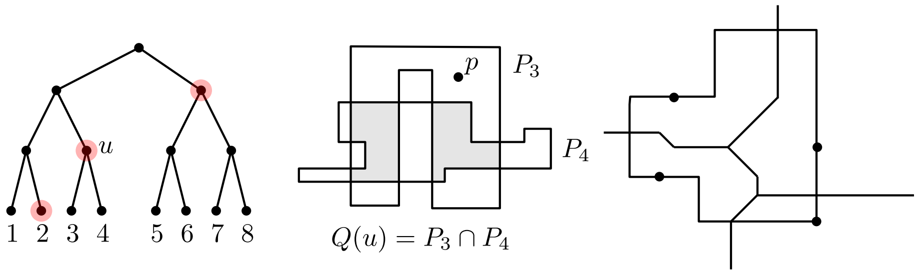

Figure 3 Left: an illustration of a segment tree over $[8]$, where the red nodes form the canonical decomposition of $[2, 8]$. Middle: the gray region depicts $Q(u)$. Right: an additively weighted Voronoi diagram on four portals under the L_1 metric.

Next we describe a subroutine that, given an index $i \in [k]$ and a point p, computes the value $\mathrm{next}(i,p)$ in time $O(\log^2 n)$. To this end, first compute the canonical decomposition of the interval $[i,k]$; that is, find $\ell \leqslant \lceil \log k \rceil$ nodes $u_1, \dots, u_\ell \in T$ such that $[i,k] = I(u_1) \cup \dots \cup I(u_\ell)$ is a disjoint union. Then compute the minimal $t \in [\ell]$ such that $p \notin Q(u_t)$. We descend from node u_t towards the leaves; at each step we go to the left child v if $p \notin Q(v)$, and the right child otherwise. In the end we arrive at a leaf that corresponds to exactly $\min\{j \geqslant i : p \notin P_i \cap \dots \cap P_j\} = \mathrm{next}(i,p)$.

Computing the canonical decomposition takes $O(\log k)$ time. Computing t needs $\ell \leqslant \log k$ queries into the point-location data structures. The descent from u_t to the leaf needs $O(\log k)$ queries as well. Since each query is answered in $O(\log n)$ time, the subroutine takes time $O(\log^2 n)$ as claimed. For each $i \in [k]$ and $p \in \mathrm{port}_i$, we call the subroutine to compute $\mathrm{next}(i,p)$ and store the values in a look-up table. Since $\sum_{i=1}^{k} |\mathrm{port}_i| \leqslant \sum_{i=1}^{k} 2n(P_i) \cdot n = 2n^2$, the table can be computed in time $O(n^2 \log^2 n)$.

With these preparations, the main algorithm is a dynamic program. As the base case, we have $f_1(p) = 0$ for all $p \in \mathrm{port}_1$. We claim the following recursive formula:

▷ Claim. For all $2 \leqslant j \leqslant k$ and $q \in \mathrm{port}_j$, we have

$$f_j(q) = \begin{cases} f_{j-1}(q) & \text{if } q \in P_{j-1}, \\ \min\{f_i(p) + \mathrm{dist}(p,q) \;:\; p \in \mathrm{port}_i,\ i<j,\ \mathrm{next}(i,p) = j\} & \text{otherwise.} \end{cases}$$

Proof. If $q \in P_{j-1}$ then $f_j(q) = f_{j-1}(q)$ by definition. From now on we assume $q \notin P_{j-1}$. The $\leqslant$ direction is easy: $f_i(p)$ corresponds to a tour visiting $P_1, \dots, P_{i-1}, p \in P_i$, and $\mathrm{dist}(p,q)$ corresponds to a tour visiting $p, P_{i+1}, \dots, P_{j-1}, q$ since $\mathrm{next}(i,p) = j$. Their sum thus corresponds to a tour visiting $P_1, \dots, P_{j-1}, q$.

For the $\geqslant$ direction, let $(i_1, q_1), \dots, (i_m, q_m)$ be a minimum length skeleton visiting $P_1, \dots, P_{j-1}, q$ given by Corollary 10. Note that $q_m = q \notin P_{j-1}$ by definition, thus $m \geqslant 2$. Moreover, we have $i_m = j$ because $q \in \bigcap_{i_m \leqslant i \leqslant j} P_i$ but $q \notin P_{j-1}$.

By the guarantee of Corollary 10, $p := q_{m-1} \in \bigcup_{i_{m-1} \leqslant i < j} \mathrm{port}_i$ and $\mathrm{next}(p, i_{m-1}) = i_m = j$. Hence there exists $i \in [i_{m-1}, j)$ such that $p \in \mathrm{port}_i$. For this i we clearly also have $\mathrm{next}(p,i) = j$. Finally, Lemma 8 implies that $f_j(q)$ is exactly the length of this skeleton, which is equal to $f_i(p) + \mathrm{dist}(p,q)$. ◁

The dynamic program iterates over $j = 2, \ldots, k$. In each iteration, we construct the additively weighted Voronoi diagram D (under L_1 metric) on the point set

$$S_j := \{p \in \text{port}_i \;:\; i < j,\ \text{next}(i, p) = j\},$$

where each point p has weight $f_i(p)$. For each $q \in \text{port}_j$, if $q \in P_{j-1}$ then we assign $f_j(q) := f_{j-1}(q)$, else we assign $f_j(q) := D.\textsc{dist}(q)$. The correctness follows from the claim above. After all iterations finish, we return $\min\{f_i(p) : p \in \text{port}_i, i \in [k], \text{next}(p, i) = k + 1\}$.

In each iteration j, constructing the Voronoi diagram takes $O(|S_j| \log n)$ time [26], and the queries take $O(|\text{port}_j| \log n)$ time. Over all iterations, the time is $O\left(\sum_{j=1}^{k}(|S_j| + |\text{port}_j|) \log n\right)$. Note that

$$\sum_{j=1}^{k} |S_j| = \sum_{j=1}^{k} \sum_{i<j} \sum_{p \in \text{port}_i} \mathbf{1}_{\text{next}(i,p)=j} = \sum_{i=1}^{k} \sum_{p \in \text{port}_i} 1 = \sum_{i=1}^{k} |\text{port}_i|.$$

Hence the total time is $O\left(\sum_{j=1}^{k} |\text{port}_j| \log n\right) = O(n^2 \log n)$. ◀

4 Touring step-disjoint orthogonal polygons

We move on to study step-disjoint orthogonal polygons $P_1, \ldots, P_k$ and prove Theorem 3, which breaks the quadratic-time barrier. As a rough outline, the algorithm decomposes the sequence of polygons into batches, where each batch either contains a small number of vertices (sparse batch), or has few polygons (dense batch). The algorithm processes the batches in order. In each round it extends the prefix shortest tours computed so far to visit one more batch. Different strategies are used for sparse and dense batches, and they are driven by different structural insights.

We will make use of two results from the literature. One is a generalized distance oracle for dynamic planar graphs [16, Theorem 18]:

▶ **Theorem 12.** *Let G be a weighted planar digraph on n nodes, with a set $U \subseteq V(G)$ of facilities. There exists a data structure maintaining G under edge insertions, edge deletions and changes of U in $\widetilde{O}(n^{3/4}|U|^{1/4} + n^{4/5})$ update time. Given any $v \in V(G)$, it can compute $\min_{u \in U} \text{dist}(u, v)$ in $\widetilde{O}(1)$ time. The initialization time is $\widetilde{O}(n)$.*

The other is a rectangle partition of $\mathbb{R}^2$ with low stabbing number [18, Lemma 3.1].

▶ **Lemma 13** (∗)**.** *In time $O(n)$, we can partition $\mathbb{R}^2$ into $n + 1$ rectangles such that*
(i) *no rectangle contains a vertex of $P_1, \ldots, P_k$ in its interior;*
(ii) *every vertical/horizontal line intersects at most $O(\sqrt{n})$ rectangles.*

4.1 Decomposition into batches

Let $0 < \alpha < \beta < 1$ be constants to be determined later. (For concreteness, think of $\alpha = 0.85$ and $\beta = 0.98$.) We will carefully choose some polygons $P_{i_1}, P_{i_2}, \ldots, P_{i_r}$ as *delimiters*, where $1 \leqslant i_1 < i_2 < \cdots < i_r \leqslant k$. They split the input sequence into *batches* $(P_1, \ldots, P_{i_1}), (P_{i_1}, \ldots, P_{i_2}), \ldots, (P_{i_r}, \ldots, P_k)$. We say that a batch $(P_i)_{a \leqslant i \leqslant b}$ is *sparse* if $\sum_{i=a}^{b} n(P_i) \leqslant 3n^{\beta}$; and it is *dense* if $b - a \leqslant n^{1-\alpha} + 1$. Note that a batch can be both sparse and dense.

▶ **Lemma 14.** *In time $O(k)$ we can compute a set of at most $3n^{1-\beta}$ delimiters, each of complexity at most n^{α}, such that every batch is either sparse or dense.*

Proof. We color each polygon P_i blue if $n(P_i) \leqslant n^\alpha$, and red otherwise. This breaks the input sequence into blue and red blocks. We further differentiate a red block $(P_i)_{a\leqslant i\leqslant b}$ by two types: it is *light red* if $\sum_{i=a}^{b} n(P_i) \leqslant n^\beta$, and *dark red* otherwise. Clearly there are at most $n^{1-\beta}$ dark red blocks.

Now we fix two consecutive dark red blocks, and consider the sequence of polygons $(P_i)_{a\leqslant i\leqslant b}$ in between. It must start with a blue block, alternate between light red and blue blocks, and end with a blue block. We mark P_a and P_b as delimiters. We then iterate over $i = a, \dots, b$ and keep a running sum N. At iteration i we let $N := N + n(P_i)$. If $N > n^\beta$ *and* P_i is blue, then we mark P_i as a delimiter and reset N to zero. We run the procedure between every pair of consecutive dark red blocks, and return all the marked delimiters. The running time is clearly $O(k)$. Next we argue that the required properties hold.

The number of delimiters is bounded as follows. Between every pair of consecutive dark red blocks, we mark (i) a delimiter in the beginning; (ii) a delimiter in the end; and (iii) one or more delimiters in the middle. Over all consecutive pairs, the total contribution of (i)(ii) is at most twice the number of dark red blocks, that is $2n^{1-\beta}$. The total contribution of (iii) is at most $n^{1-\beta}$ because each time we mark a delimiter this way, the running sum must have exceeded n^β.

By construction every delimiter is blue (i.e., has complexity at most n^α). Now consider an arbitrary batch $(P_i)_{a\leqslant i\leqslant b}$. There are only two cases:

- It consists of a dark red block and two adjacent blue polygons. We have $b - a \leqslant n^{1-\alpha} + 1$ since each red polygon has complexity at least n^α and the total complexity is at most n. Hence the batch is dense.
- It is formed when we scan the sequence between consecutive dark red blocks. The scanning procedure ends the batch as soon as the running sum N exceeds n^β and we see a blue polygon. Since each light red block and each blue polygon can contribute at most n^β to the sum, we have $\sum_{i=a}^{b} n(P_i) \leqslant 3n^\beta$. Hence the batch is sparse. ◀

From now on we assume that the decomposition in Lemma 14 is computed. Our algorithm will process the batches in order and pass necessary information from one batch to the next. When we finish processing a batch $(P_i)_{a\leqslant i\leqslant b}$, we would have computed $f_b(p)$ for all $p \in \mathrm{port}_b$. We employ different strategies to process sparse and dense batches. Let us explain the high-level ideas.

Inside a sparse batch, we define *terminals* as the local analogue of portals. More precisely, these are the boundary points on the grid induced by the batch. Clearly there are at most $(3n^\beta)^2 = 9n^{2\beta}$ terminals. We show that there exists a global shortest tour that traverses this batch in one of three ways: (i) it visits a terminal on each polygon in the batch; (ii) it travels horizontally throughout the batch; or (iii) it travels vertically throughout the batch. To handle (i), we use a simple dynamic program over terminals, which costs only polylogarithmic time per terminal. To handle (ii) or (iii), we build a dynamic planar graph and use a line sweep procedure to extract shortest tours to all portals on the last polygon.

Inside a dense batch, the number of terminals can be quadratic, so (i) can no longer be handled efficiently even though the structural insight remains valid. To overcome this barrier, we partition the plane into rectangles with a low stabbing number, which effectively summarizes the geometry. This way, we obtain only $O(n^{3/2})$ (instead of $O(n^2)$) points worth considering in the dynamic program, which we call *hubs*. Since the number of polygons is small in a dense batch, this already implies a subquadratic-time dynamic program. However, a hub might not lie on the boundary of any polygon, so we have to deal with the intricate problem of converting f-values on portals to f-values on hubs, and vice versa. We apply two line sweeps based on dynamic planar graphs, one before and one after the dynamic program, to convert between the two worlds.

4.2 Processing sparse batches

▶ **Theorem 15.** *Let $(P_i)_{a \leqslant i \leqslant b}$ be a sparse batch. Given $f_a(p)$ for all $p \in \mathrm{port}_a$, we can compute $f_b(p)$ for all $p \in \mathrm{port}_b$ in time $\widetilde{O}(n^{2\beta} + n^{1+\alpha} + n^{(7+\alpha)/4} + n^{9/5})$.*

We devote the section to proving Theorem 15. Throughout we fix a sparse batch $(P_i)_{a \leqslant i \leqslant b}$. We define the *terminals* of P_i as $\mathrm{term}_i := \partial P_i \cap \mathrm{grid}(\{P_i\}_{a \leqslant i \leqslant b})$. Note that $\mathrm{term}_i \subseteq \mathrm{port}_i$. Since a sparse batch has complexity $O(n^\beta)$, the number of terminals is $O(n^{2\beta})$.

▶ **Lemma 16.** *Let $p \in \mathrm{port}_b$. Among all shortest tours visiting $P_1, \dots, P_{b-1}, p$, there is a tour that satisfies one of the following properties:*

1. *it visits* $\mathrm{port}_1, \dots, \mathrm{port}_{a-1}, \mathrm{term}_a, \dots, \mathrm{term}_{b-1}, p$*;*
2. *it visits* $\mathrm{port}_1, \dots, \mathrm{port}_a$ *and stays on a horizontal line afterwards until it reaches* p*;*
3. *it visits* $\mathrm{port}_1, \dots, \mathrm{port}_a$ *and stays on a vertical line afterwards until it reaches* p*.*

Proof. Let π be a shortest tour visiting $P_1, \dots, P_{b-1}, p$ given by Corollary 11. We claim that π is the tour we are looking for.

To this end, define $p_1 \in \mathrm{port}_1, \dots, p_{b-1} \in \mathrm{port}_{b-1}$ as in Corollary 11. First suppose that none of $p_a, \dots, p_{b-1}$ is a vertex. If p_{b-1} is on a horizontal edge (thus not on a vertical edge), then Corollary 11 guarantees that $x(p_{b-2}) = x(p_{b-1}) = x(p)$ and p_{b-2} is on a horizontal edge as well. Propagating the argument in reverse order of time, we conclude that $x(p_a) = \cdots = x(p_{b-1}) = x(p)$, so π satisfies property 3. In the symmetric case that p_b is on a vertical edge, we conclude that π satisfies property 2.

Next suppose that some of $p_a, \dots, p_{b-1}$ are vertices. We split the sequence $p_a, \dots, p_{b-1}$ at these vertices into subsequences. Consider any subsequence but the first one. It must start from a vertex, say p_i, and does not contain any other vertex. So either all points in it share the same x-coordinate $x(p_i)$, or all share the same y-coordinate $y(p_i)$. Hence all are terminals. For the first subsequence, we can make the same argument from where it ends. We have thus shown that $p_i \in \mathrm{term}_i$ for all $a \leqslant i < b$, hence π satisfies property 1. ◀

For index $a \leqslant i \leqslant b$ and point $p \in \mathrm{port}_i$, let $f_i^1(p)$ be the minimum length among all tours that visit $\mathrm{port}_1, \dots, \mathrm{port}_{a-1}, \mathrm{term}_a, \dots, \mathrm{term}_{i-1}, p$. Let $f_i^2(p)$ be the minimum length among all tours that visit $\mathrm{port}_1, \dots, \mathrm{port}_a, P_{a+1}, \dots, P_{i-1}, p$ and stay horizontal after port_a. Let $f_i^3(p)$ be defined similarly to $f_i^2(p)$ but now the tour must stay vertical after port_a. Lemma 16 states that $f_i(p) = \min\{f_i^1(p), f_i^2(p), f_i^3(p)\}$, so we compute the three values separately.

Type-1 tours. For $a \leqslant i \leqslant b$ and $p \in \mathrm{port}_i$, we have the recursive formula

$$f_i^1(p) = \begin{cases} f_i(p) & i = a, \\ \min\left\{f_{i-1}^1(q) + \mathrm{dist}(p, q) : q \in \mathrm{term}_{i-1}\right\} & i \in (a, b]. \end{cases}$$

This leads to a straightforward dynamic program: For $i = a+1, \dots, b$, build a Voronoi diagram D over all $q \in \mathrm{term}_{i-1}$ using $f_{i-1}^1(q)$ as additive weights, then let $f_i^1(p) := D.\textsc{dist}(p)$ for all $p \in \mathrm{term}_i$. (In the last iteration we do so for all $p \in \mathrm{port}_b$.)

Let us analyze the running time of iteration i. Building the Voronoi diagram takes time $\widetilde{O}(|\mathrm{term}_{i-1}|)$. Querying the Voronoi diagram takes time $\widetilde{O}(|\mathrm{term}_i|)$ in total. (For the last iteration querying takes time $\widetilde{O}(|\mathrm{port}_b|)$, which is $\widetilde{O}(n^{1+\alpha})$ because the delimiter P_b has complexity n^α.) Recall that $\sum_{i=a}^b |\mathrm{term}_i| = O(n^{2\beta})$ by sparsity, the time complexity over all iterations is $\widetilde{O}(n^{2\beta} + n^{1+\alpha})$.

Type-2/3 tours. These two types are symmetric, so we focus on type 2. We sort $\mathcal{Y} := y(P_1) \cup \cdots \cup y(P_k)$ in increasing order and cut it into contiguous groups of size $n^{1-\alpha}$ each.

Fix an arbitrary group $Y \subset \mathcal{Y}$. We sweep a horizontal line $y \in Y$ bottom up. At each sweep step we aim to compute $f_2(p)$ for all $p \in \mathrm{port}_b \cap y$. To this end we build a weighted planar digraph $G^{(y)}$ as follows (see Figure 4 for an illustration).

- The node set consists of multiple layers $S \cup X_a \cup \cdots \cup X_b$, where

$$X_i := \begin{cases} x(P_i) & i = a \\ x(P_i) \cup x(P_{i-1}) & a < i \leqslant b \end{cases}$$

$$S := \{s_{x,h} \ : \ x \in X_a \ , h \in Y\}.$$

  The nodes in each layer X_i are arranged on a line in the natural order. A node $x \in X_i$ is *active* if (x, y) is a portal.
- For every pair of neighboring nodes $x, x' \in X_i$ in the same layer, we add an edge $x \leftrightarrow x'$ of weight $|x - x'|$.
- For every active node $x \in X_i$ and its copy $x' \in X_{i+1}$, we add an edge $x \to x'$ of weight 0.
- For every $s_{x,h} \in S$, we add an edge $s_{x,h} \to x$. The weight is $f_a(x, h)$ if (x, h) is a portal, and ∞ otherwise.
- Finally, we specify $U^{(y)} := \{s_{x,y} : x \in X_a\}$ as the facilities.

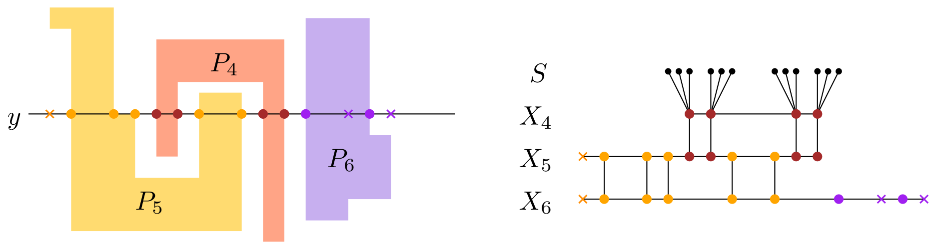

Figure 4 The structure of the plane graph $G^{(y)}$ for a batch of three polygons P_4, P_5, P_6. Active nodes are drawn as dots, and inactive nodes are drawn as crosses. Edges are oriented downwards, whose weights are omitted for clarity.

It is clear from the construction that $G^{(y)}$ is planar. Note that $|S| = O(n^\alpha \cdot |Y|) = O(n)$, and $\sum_{i=a}^{b} |X_i| \leqslant 2 \sum_{i=a}^{b} n(P_i) = O(n)$ by sparsity. So the number of nodes is bounded by $O(n)$. Moreover, we have $f_2(x, y) = \min_{u \in U^{(y)}} \mathrm{dist}(u, x)$ for all $x \in X_b$. In other words, the f^2-values can be retrieved by querying the corresponding nodes in the graph.

Observe that $G^{(y)}$ changes marginally when we sweep y for one step: The set of nodes do not change, and at most two inter-layer edges are inserted/removed due to activation/de-activation of nodes. The facilities $U^{(y)}$ change completely, but the size is bounded by $|x(P_b)| \leqslant n^\alpha$.

Applying the data structure from Theorem 12, we get the following running time:

- Initialization takes time $\widetilde{O}(n)$;
- In each sweep step, it takes time $\widetilde{O}(n^{(3+\alpha)/4} + n^{4/5})$ to update the graph, and time $\widetilde{O}(n^\alpha)$ to query the distances to portals.

Finally, we sum over all groups Y. There are n^α groups, thus the same number of initializations. Over all groups there are $O(n)$ sweep steps. So the total time of handling type-2 tours in this batch is

$$\widetilde{O}\left(n^{1+\alpha} + n^{1+(3+\alpha)/4} + n^{1+4/5} + n^{1+\alpha}\right) = \widetilde{O}\left(n^{1+\alpha} + n^{(7+\alpha)/4} + n^{9/5}\right).$$

Summary. Having computed f_b^1, f_b^2, f_b^3, we can compute $f_b(p) = \min\{f_b^1(p), f_b^2(p), f_b^3(p)\}$ for all $p \in \mathrm{port}_b$. Correctness is guaranteed by Lemma 16. This completes the proof of Theorem 15.

4.3 Processing dense batches

▶ **Theorem 17.** *Let $(P_i)_{a \leqslant i \leqslant b}$ be a dense batch. Given $f_a(p)$ for all $p \in \mathrm{port}_a$, we can compute $f_b(p)$ for all $p \in \mathrm{port}_b$ in time $\widetilde{O}(n^\gamma)$, where $\gamma := \max\{\frac{9}{5}, 1+\alpha, \frac{7+\alpha}{4}, \frac{5}{2}-\alpha, \frac{7}{2}-2\alpha, \frac{9-2\alpha}{4}, \frac{33-2\alpha}{16}\}$.*

We devote the section to proving Theorem 17. Throughout we fix a dense batch $(P_i)_{a \leqslant i \leqslant b}$, which by definition satisfies $b-a = O(n^{1-\alpha})$. We also fix a rectangle partition from Lemma 13. Property (i) in Lemma 13 implies that for any polygon P_i and rectangle R, the intersection $P_i \cap R$ is either a collection of vertical stripes, a collection of horizontal stripes, or empty.[1] If it is non-empty, we say that P_i is *vertical* or *horizontal* in R, respectively.

A *hub* is a point $p \in \mathrm{grid}(\{P_1, \ldots, P_k\})$ that appears on the boundary of some rectangle. Equivalently, every hub is on the intersection of some vertical/horizontal line through the polygon vertices and some rectangle boundary. Since every line may generate at most $O(\sqrt{n})$ hubs by property (ii) in Lemma 13, and since there are $2n$ lines in total, we have the following:

▶ **Observation 18.** *The number of hubs is $O(n^{3/2})$.*

Before we continue, we introduce a canonical decomposition of a tour into phases. Assume that tour π visits $p_a, \ldots, p_b$ in sequence, where p_i is the first point in P_i visited by π. Let $q_1, \ldots, q_h$ be the sequence of hubs visited by π. A subtour $\pi[q_a, q_{a+1}]$ between consecutive hubs is called a *progression phase* if it contains some point p_i where $a \leqslant i \leqslant b$ (generally it may contain a sequence of points $p_i, \ldots, p_j$). The subtours sandwiched between progression phases are called *teleportation phases*. See Figure 5 for an example.

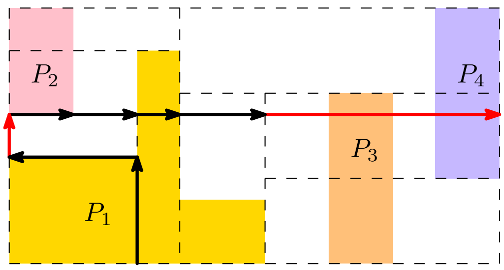

Figure 5 An illustration of a tour that visits four polygons P_1, P_2, P_3, P_4 and nine hubs. It has two progression phases (in red) and two teleportation phases (in black).

▶ **Lemma 19.** *Among all shortest tours that visit $P_1, \ldots, P_b$ in sequence, there is a tour such that every progression phase is confined within a rectangle of the decomposition.*

Proof. We take a shortest tour π that conforms to Corollary 11. Recall that π snaps to $\mathrm{grid}(\{P_1, \ldots, P_k\})$, its intersection points with the rectangle boundaries are hubs. Recall that every progression phase of π is a subtour between consecutive hubs, so it must be confined within a rectangle. ◀

[1] Technically, P_i and R might intersect only on the boundary of R. Nevertheless, this corner case does not affect our arguments. Our algorithm still works by considering the adjacent rectangle R' for which $P_i \cap R'$ is indeed either a collection of vertical stripes or a collection of horizontal stripes.

Zooming into a progression phase, we show that vertical and horizontal movements are in a sense independent.

▶ **Lemma 20.** *Let R be a rectangle from the decomposition. Consider a shortest tour and its subtour π inside R. Suppose that π starts at hub p, visits $P_i, \ldots, P_j$ in sequence, and ends at hub q. Let $I_{\mathrm{h}} \subseteq \{i, \ldots, j\}$ be the indices corresponding to horizontal polygons in R, and $I_{\mathrm{v}} \subseteq \{i, \ldots, j\}$ be the indices corresponding to vertical polygons in R. Then the vertical movements of π, all combined, is a shortest vertical tour that visits $x(p), (P_t)_{t\in I_{\mathrm{h}}}, x(q)$ in R. Similarly, the horizontal movements of π, all combined, is a shortest horizontal tour that visits $y(p), (P_t)_{t\in I_{\mathrm{v}}}, y(q)$ in R.*

Proof. We prove the statement for vertical movements; the proof for horizontal movements is symmetric. Assume towards contradiction that there is a shorter vertical tour η that visits $x(p), (P_t)_{t\in I_{\mathrm{h}}}, x(q)$ in R. We replace the vertical movement of π that reaches a polygon P_i by the movement of η that reaches P_i, without changing the order of the movements. The resulting tour starts at hub p, visits $P_i, \ldots, P_j$ in sequence, ends at hub q and is shorter, contradicting the optimality of π. ◀

Our algorithm mirrors the alternation between progression and teleportation phases: In a progression phase we walk inside a rectangle and visit a sequence of new polygon(s), while in a teleportation phase we may jump far in space but do not visit any new polygon. The algorithm can be outlined as follows:

- We use the known $f_a(\cdot)$ values on portals of P_a to initialize $f_a(\cdot)$ on hubs.
- We run a dynamic program that iterates over $j = a+1, \ldots, b$. In each iteration we compute $f_j(\cdot)$ by implementing one alternation of progression and teleportation phases.
- In the end, all hubs receive the correct $f_b(\cdot)$ values. We project them back to $f_b(\cdot)$ values on portals of P_b, thereby achieving the goal of the batch.

We will now elaborate on the algorithm.

Initialization. For initialization, we want to derive $f_a(\cdot)$ values on hubs from the already computed $f_a(\cdot)$ values on port_a. We use the same line-sweep procedure as we did for type-2 and 3 tours in sparse batches. Namely, we sweep a global line y and maintain a dynamic planar graph $G^{(y)}$. The graph is unchanged, so the time analysis for updating edges/facilities remains valid. On the other hand, the queries are slightly different: For every hub $p = (x, y)$ and every index $a \leqslant i \leqslant b$, we initialize $f_i(p) := \min\{\textsc{query}(x') + |x' - x|, \textsc{query}(x'') + |x'' - x|\}$, where (x', y) is the closest portal on P_i to the left of p, and (x'', y) is the closest portal on P_i to the right of p. Hence, for every sweep step we make $O((b-a)\sqrt{n}) = O(n^{3/2-\alpha})$ queries. Summing over $O(n)$ sweep steps, the total query time is $\widetilde{O}(n^{5/2-\alpha})$.

Altogether, the initialization costs time $\widetilde{O}(n^{1+\alpha} + n^{(7+\alpha)/4} + n^{5/2-\alpha} + n^{9/5})$.

Dynamic program. For each $a \leqslant j \leqslant b$, rectangle R and hub $q \in \partial R$, we define $g_j(R, q)$ as the minimum length among all tours that visit $P_1, \ldots, P_j, q$ and whose final leg is a progression phase in R. The next two lemmas relate f and g.

▶ **Lemma 21.** *Fix $a \leqslant j \leqslant b$. Given $g_j(R, p)$ for all rectangles R and hubs $p \in \partial R$, we can compute $f_j(q)$ for all hubs q in time $\widetilde{O}(n^{3/2})$.*

Proof. Recall that $f_j(q)$ is the minimum length of a tour that visits $P_1, \ldots, P_j, q$. Note that

$$f_j(q) = \min_{\text{hub } p} \left(\operatorname{dist}(p, q) + \min_{R \ni p} g_j(R, p) \right).$$

The $\leqslant$ direction is clear since the right hand side always composes a tour that visits $P_1, \ldots, P_j, q$ in sequence. For the $\geqslant$ direction, consider a tour defining $f_j(q)$. If its last leg was a progression phase in some R, then by definition $f_j(q) = g_j(R, q)$. Else, its last leg was a teleportation phase from some hub p, prior to which was a progression phase in some R. So $f_j(q) = g_j(R, p) + \text{dist}(p, q)$.

To compute it efficiently, we build a Voronoi diagram D on hubs, with each hub p additively weighted by $\min_{R \ni p} g_j(R, p)$. Then for each hub q, we compute $f_j(q) := D.\text{DIST}(q)$. Correctness follows directly from the recursion above, so it remains to analyze time complexity. The Voronoi diagram has $O(n^{3/2})$ sites (hubs) by Observation 18. Computing the additive weight of each site takes time $O(1)$, so the construction of the diagram takes time $\widetilde{O}(n^{3/2})$. Every hub gives rise to one query into the diagram, which takes time $\widetilde{O}(1)$. So the total query time amounts to $\widetilde{O}(n^{3/2})$. ◀

▶ **Lemma 22.** *Fix R and $a \leqslant j \leqslant b$. Given $f_a(p), \ldots, f_{j-1}(p)$ for all hubs $p \in \partial R$, we can compute $g_j(R, q)$ for all hubs $q \in \partial R$ in time $O(n^{1-\alpha} N \log N)$, where N is the number of hubs on ∂R.*

Proof. Write $i \preceq j$ if $a \leqslant i \leqslant j \leqslant b$ and each of $P_i, \ldots, P_j$ intersects R. In this case, we can partition $\{i, \ldots, j\} =: I_\text{v} \cup I_\text{h}$ based on whether a polygon is vertical or horizontal in R. Let $\text{hcost}(R, i, j, x, x')$ be the length of the shortest horizontal tour that visits $x, (P_t)_{t \in I_\text{v}}, x'$ in R; let $\text{vcost}(R, i, j, y, y')$ be the length of the shortest vertical tour that visits $y, (P_t)_{t \in I_\text{h}}, y'$ in R. We claim that

$$g_j(R, q) = \min_{\substack{\text{hub } p \in \partial R \\ i \preceq j}} \big(f_{i-1}(p) + \text{hcost}(R, i, j, x(p), x(q)) + \text{vcost}(R, i, j, y(p), y(q))\big).$$

The $\leqslant$ direction is clear. For the $\geqslant$ direction, we consider a shortest tour that defines $g_j(R, q)$. Its final leg is a progression phase in R that visits a non-empty sequence of polygons $P_i, \ldots, P_j$ where $a \leqslant i \leqslant b$ and $i \preceq j$. This part of the tour may contain horizontal and vertical movements, and by Lemma 20 the two movements have length $\text{hcost}(R, i, j, x(p), x(q))$ and $\text{vcost}(R, i, j, y(p), y(q))$, respectively. Before this, the tour must have visited $P_1, \ldots, P_{i-1}$, so it has length $f_{i-1}(p)$ by definition.

We will now compute the recursion efficiently. For each edge $e \subset R$, we define

$$g_j^e(R, q) := \min_{\substack{\text{hub } p \in e \\ i \preceq j}} \big(f_{i-1}(p) + \text{hcost}(R, i, j, x(p), x(q)) + \text{vcost}(R, i, j, y(p), y(q))\big).$$

Clearly $g_j = \min\big\{g_j^e : e \text{ is an edge of } R\big\}$. So it remains to compute g_j^e for each e.

By symmetry we focus on the case that e is the left edge. Since $x(p) = x(e)$ is a fixed value for all hubs $p \in e$, we can factor out the horizontal cost and write

$$g_j^e(R, q) = \min_{i \preceq j} \left(\text{hcost}(R, i, j, x(e), x(q)) + \min_{\text{hub } p \in e} (f_{i-1}(p) + \text{vcost}(R, i, j, y(p), y(q)))\right).$$

From now on we fix $i \preceq j$. We compute $\text{hcost}(R, i, j, x(e), x(q))$ for all q by running Dijkstra's algorithm on a layered graph. Specifically, let $L \subseteq \{i, \ldots, j\}$ be the indices of the vertical polygons in R. For each $\ell \in L$ we define a layer $V_\ell := x(P_\ell \cap R)$. We also introduce a source layer $V_0 := \{x(e)\}$ as well as a sink layer V_∞ that contains the x-coordinates of hubs on R. The edges are defined as follows:

- We arrange the nodes in each layer on a real line, and add an edge between every pair of adjacent nodes. The edge weight is the difference of their x-coordinates.

- For $\ell \in \{0\} \cup L$ and every $x \in V_\ell$, we find the smallest index $\ell' > \ell$ such that $P_{\ell'} \cap R$ does *not* cover x. (We set $\ell' := \infty$ if no such index exists.) Then we add an edge from x to
  - the closest node $x' \in V_{\ell'}$ to the left of x; and
  - the closest node $x' \in V_{\ell'}$ to the right of x.

  The edge weights are both $|x - x'|$.

By construction, a shortest tour from $x(e)$ to $x \in V_\infty$ in the graph corresponds to a shortest horizontal tour that visits $x(e), (P_\ell)_{\ell \in L}, x$ in R. So to compute $\mathrm{hcost}(R, i, j, x(e), x(q))$ for all hubs q, it suffices to run Dijkstra's algorithm from $x(e)$ and read off the distances to all $x \in V_\infty$. Note that the number of nodes is bounded by $O(N)$, and the number of edges is linear in the number of nodes. Hence Dijkstra's algorithm runs in time $O(N \log N)$.

In a similar fashion we compute $\min_{p \in e}(f_{i-1}(p) + \mathrm{vcost}(R, i, j, y(p), y(q)))$ for all q. This time, let $L \subseteq \{i, \dots, j\}$ collect the indices of the horizontal polygons in R. For each $\ell \in L$ we define a layer $V_\ell := y(P_\ell \cap R)$. We introduce a source node s and two more layers V_0, V_∞. The layer V_0 contains the y-coordinates of hubs on e. The layer V_∞ contains the y-coordinates of all hubs on R. The edges are defined as follows:

- We add an edge from the source s to every node $y \in V_0$ with weight $f_{i-1}(x(e), y)$.
- We arrange the nodes in each layer on a real line, and add an edge between every pair of adjacent nodes. The edge weight is the difference of their y-coordinates.
- For $\ell \in \{0\} \cup L$ and every $y \in V_\ell$, we find the smallest index $\ell' > \ell$ such that $P_{\ell'} \cap R$ does *not* cover y. (We set $\ell' := \infty$ if no such index exists.) Then we add an edge from y to
  - the closest node $y' \in V_{\ell'}$ above y; and
  - the closest node $y' \in V_{\ell'}$ below y.

  The edge weights are both $|y - y'|$.

Then we run Dijkstra's algorithm from s and read off the distances to all $y \in V_\infty$. The running time is again $O(N \log N)$.

After we have computed the two terms, we add them for each q. We iterate the procedure for every $i \preceq j$ and thereby obtain g_e, from which we compute g. Since there are at most $b - a = O(n^{1-\alpha})$ iterations, the total running time is as claimed. ◀

▶ **Corollary 23.** *Fix $a \leqslant j \leqslant b$. Given $f_a(p), \dots, f_{j-1}(p)$ for all hubs $p \in \partial R$, we can compute $g_j(R, q)$ for all rectangles R and hubs $q \in \partial R$ in time $\widetilde{O}(n^{5/2-\alpha})$.*

Proof. We apply Corollary 23 for each rectangle R. Since each hub appears on at most four rectangles, the sum over all N's is bounded by four times the number of hubs, which is $O(n^{3/2})$. ◀

Now we can assemble the dynamic program. For $j = a+1, \dots, b$, we apply Corollary 23 to compute $g(\cdot, \cdot, j)$ from $f_a(\cdot), \dots, f_{j-1}(\cdot)$, then we apply Lemma 21 to compute $f_j(\cdot)$ from $g(\cdot, \cdot, j)$. Hence, each iteration corresponds to an alternation of progression and teleportation phases. When the last iteration finishes, we obtain $f_b(q)$ for all hubs q.

Since the number of iterations is $b - a = O(n^{1-\alpha})$, the time complexity is bounded by $\widetilde{O}(n^{1-\alpha} \cdot (n^{3/2} + n^{5/2-\alpha})) = \widetilde{O}(n^{7/2-2\alpha})$.

Projection. As the final step, we project the computed f_b-values on hubs to f_b-values on portals. Not surprisingly, we apply yet another line sweep, but the details are quite different from initialization.

Let $\mathcal{Y} := \bigcup_{i=1}^{k} y(P_i)$. We sort $\mathcal{Y}$ in increasing order and break it into contiguous groups of size $n^{(2\alpha-1)/4}$ each. Now fix an arbitrary group $Y \subset \mathcal{Y}$. We sweep a horizontal line $y \in Y$ bottom up. Since each y hits $O(\sqrt{n})$ hubs, there are at most $O(n^{(2\alpha+1)/4})$ relevant hubs for the group. We collect the x-coordinates of these hubs into a set X.

For each line y, we build a weighted planar digraph $G^{(y)}$ as follows:

- The node set consists of multiple layers $S \cup X_a \cup \cdots \cup X_b$, where

$$X_i := \begin{cases} X \cup x(P_i) & i = a \\ X \cup x(P_i) \cup x(P_{i-1}) & a < i \leqslant b \end{cases}$$
$$S := \{s_{i,x,h} \ : \ a \leqslant i \leqslant b, \ x \in X \ , h \in Y\}.$$

  The nodes in each layer X_i are arranged on a line in the natural order. A node $x \in X_i$ is *active* if (x, y) is a portal.
- For every pair of neighboring nodes $x, x' \in X_i$ in the same layer, we add an edge $x \leftrightarrow x'$ of weight $|x - x'|$.
- For every active node $x \in X_i$ and its copy $x' \in X_{i+1}$, we add an edge $x \to x'$ of weight 0.
- For every $s_{i,x,h} \in S$, we add an edge $s_{i,x,h} \to x$. The weight is $f_i(x, h)$ if (x, h) is a hub, and ∞ otherwise.
- Finally, we specify $U^{(y)} := \{s_{i,x,y} : a \leqslant i \leqslant b, x \in X\}$ as the facilities.

See Figure 6 for an illustration. Clearly the graph is planar. Since the complexity of this batch is at most $O(n)$, and $b - a = O(n^{1-\alpha})$, we can bound $|G^{(y)}| = O(n + n^{1-\alpha} \cdot |X| \cdot |Y|) = O(n)$ and $|U^{(y)}| = O(n^{1-\alpha} \cdot |X|) = O(n^{(5-2\alpha)/4})$.

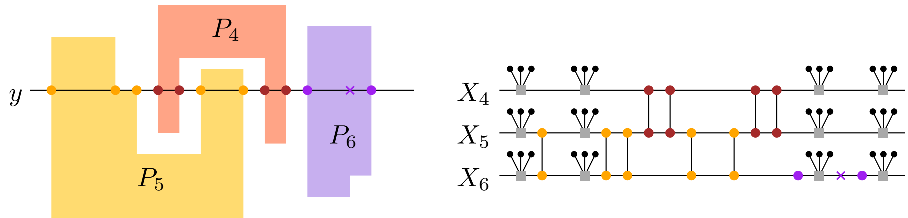

Figure 6 The structure of the plane graph $G^{(y)}$ for a batch of three polygons P_4, P_5, P_6. Active nodes are drawn as dots, and inactive nodes are drawn as crosses. The nodes that correspond to X are drawn as squares. Edges are oriented downwards, whose weights are omitted for clarity.

When we sweep y for one step, the graph $G^{(y)}$ changes only marginally. The set of nodes do not change, and at most two edges are inserted/removed due to activation/deactivation of nodes. The facilities $U^{(y)}$ change completely. To retrieve $f_b(p)$ for every $p = (x, y) \in \text{port}_b \cap y$, we simply query the distance from $U^{(y)}$ to $x \in X_b$. The data structure in Theorem 12 takes $\widetilde{O}(n)$ time at initialization. For each sweep step, it takes time $\widetilde{O}(n^{(17-2\alpha)/16})$ to update the graph and facilities, and time $\widetilde{O}(n^{\alpha})$ to query the distance to portals.

Finally, we sum over all groups Y. There are $n/|Y| = n^{(5-2\alpha)/4}$ groups, thus the same number of initializations. Over all groups there are $O(n)$ sweep steps. So the total time is

$$\widetilde{O}\left(n^{1+(5-2\alpha)/4} + n^{1+(17-2\alpha)/16} + n^{1+\alpha}\right) = \widetilde{O}\left(n^{(9-2\alpha)/4} + n^{(33-2\alpha)/16} + n^{1+\alpha}\right).$$

Summary. Theorem 17 follows by chaining the initialization, dynamic program and projection.

4.4 Wrapping up the proofs of Theorems 3 and 4

Having shown our batching strategy and the way both sparse and dense batches are handled, we are now ready to prove our main result.

▶ **Theorem 3** (Main theorem). *Orthogonal Polygon Touring(n,k) for step-disjoint orthogonal polygons can be solved in $\widetilde{O}(n^{2-\frac{1}{48}})$ time.*

Proof. We first compute a decomposition into sparse and dense batches by Lemma 14, which takes time $O(k)$. If the first batch is sparse we can set $f_1(p) = 0$ for all portals $p \in \mathrm{port}_1$. Because the batch is sparse there are at most $O(n^{1+\beta})$ of portals on P_1. If the first batch is dense we alter its initialization procedure slightly: We use a single source node instead of a source layer in the graph $G^{(y)}$, and all edges incident to the source have weight zero; during the line sweep we do not update the facilities. The time bound remains valid. Then we chain Theorems 15 and 17 to process the batches in order. Each batch requires time $\widetilde{O}(n^{\gamma})$ where

$$\gamma := \max\left\{\frac{9}{5},\ 2\beta,\ 1+\alpha,\ \frac{7+\alpha}{4},\ \frac{5}{2}-\alpha,\ \frac{7}{2}-2\alpha,\ \frac{9-2\alpha}{4},\ \frac{33-2\alpha}{16}\right\}.$$

If the last batch is sparse we return the minimum of $f_k(p)$ over all $p \in \mathrm{port}_k$. This gives the shortest tour length by Corollary 11. Since the batch is sparse, the time is bounded by $O(|\mathrm{port}_k|) = O(n^{1+\beta})$. If the last batch is dense we alter the projection procedure slightly. Instead of making a layer with portals in the graph $G^{(y)}$, we create a sink node and add an edge from each node in the last layer to the sink. The minimum tour length is exactly the distance from the source layer to the sink. The time bound remains valid. If we are interested in reconstructing the actual tour, we can use standard backtracking in the dynamic program.

We choose parameters $\alpha := 5/6 < 47/48 =: \beta$, thus $\gamma = 47/24$. Since there are at most $3n^{1-\beta} = O(n^{1/48})$ batches, the entire algorithm runs in time $\widetilde{O}(n^{95/48}) = \widetilde{O}(n^{2-\frac{1}{48}})$. ◀

Our handling of the dense cases yields a proof of Theorem 4 as a byproduct.

▶ **Theorem 4.** *Orthogonal Polygon Touring(n,k) can be solved in $\widetilde{O}(n^{1.5}k^2)$ time.*

5 Touring step-disjoint ortho-convex polygons

In this section we study step-disjoint *ortho-convex* polygons. A polygon P is ortho-convex if $P \cap \ell$ is a (possibly empty) line segment for every horizontal and vertical line ℓ. We show the following theorem.

▶ **Theorem 5.** *Orthogonal Polygon Touring(n,k) can be solved in $O(n \log n)$ time for step-disjoint ortho-convex polygons.*

Our algorithm iterates over $P_1, \ldots, P_k$. In iteration $i \in [k]$, the goal is to compute for every edge $e \subset \partial P_i$ the restriction f_i^e of f_i on e. After all iterations, we simply output the minimum of $f_k^e(p)$ over all edges $e \subset \partial P_k$ and portals $p \in e$. The correctness is guaranteed by Corollary 11.

Here is the challenge: In the worst case the total complexity of f_i over all $i \in [k]$ is $\Theta(n^2)$, so we cannot explicitly compute them all. We bypass the issue by representing them in some (interrelated) data structures which avoid explicit evaluation on too many points.

In Section 5.1, we formalize the interface and guarantee of the data structure. Then in Section 5.2, we explain how to implement the algorithm efficiently by applying the data structure as a blackbox. Finally, the full version of this paper includes an implementation of the data structure itself.

5.1 Dynamic mountain ranges

We call a continuous, piecewise linear function $f : [a,b] \to \mathbb{R}$ a *mountain range* if every piece has slope 0, -1 or 1. Its *complexity* $|f|$ is the number of pieces it contains. A *dynamic* mountain range f is a mountain range that is initially zero everywhere and undergoes the following updates:

- RESTRICT(a,b): restrict the domain to $[a,b]$.
- SHIFT(δ): add $\delta \in \mathbb{R}$ to f pointwise.
- RELAX(λ, γ): given $\lambda \in \{-1, 1\}$ and $\gamma \in \mathbb{R}$, replace f by f' where

$$f'(x) := \min\{f(x),\ \lambda x + \gamma\}$$

- JOIN(g): denote the domain of f by $[a,b]$. Given another mountain range g over domain $[b,c]$ such that $f(b) = g(b)$, replace f by f' where

$$f'(x) := \begin{cases} f(x) & x \in [a,b], \\ g(x) & x \in [b,c]. \end{cases}$$

It also supports the query operation EVALUATE(x): given any x in the domain, report $f(x)$.

▶ **Lemma 24** ($*$). *There is a fully-persistent data structure that maintains a dynamic mountain range f in amortized time $O(\log |f|)$ per operation.*

As a remark, *persistence* is needed in our application because we sometimes want to make parallel updates to the same mountain range and keep all the results.

5.2 The algorithm

The notion of (dynamic) mountain ranges is motivated by the following consideration. Corollary 11 implies that for each $i \in [k]$ the function f_i^e is a mountain range whose pieces meet at portals; moreover, $\sum_{e \subset \partial P_i} |f_i^e| = O(n)$ due to ortho-convexity. Note that in the worst case $\sum_{i=1}^{k} \sum_{e \subset P_i} |f_i^e|$ can be $\Theta(n^2)$, so we cannot afford to represent all of them explicitly. The trick is to break down these long mountain ranges into fragments that are interrelated by the shift/relax operations. In this way, we only need to represent a small subset of fragments by the dynamic mountain range data structure and infer the others when requested.

More specifically, our algorithm initializes a dynamic mountain range f_1^e for every edge $e \subset \partial P_1$, using Lemma 24. Now focus on iteration $i \in [2,k]$. We inductively assume that the previous iteration has computed f_{i-1}^e for all edges $e \subset \partial P_{i-1}$. We call $p \in \partial P_{i-1} \cup \partial P_i$ a *terminal* if $p \in \operatorname{grid}(\{P_{i-1}, P_i\})$. Note that a terminal is a portal, but not vice versa. We split every edge of P_{i-1}, P_i at terminals, resulting in *fragments*. There are at most $O(n(P_{i-1}) + n(P_i))$ terminals (and thus fragments) because each vertical/horizontal line hits the boundary of an ortho-convex polygon at most twice. It is straightforward to obtain f_{i-1}^T for all fragments $T \subset \partial P_{i-1}$ using RESTRICT. Based on these dynamic mountain ranges, the next two lemmas allow us to compute f_i^S for all fragments $S \subset \partial P_i$ efficiently. Finally, we can piece together f_i^e for all edges $e \subset \partial P_i$ using JOIN, and finish the goal of iteration i.

▶ **Lemma 25.** *Given dynamic mountain ranges f_{i-1}^T for all fragments $T \subset \partial P_{i-1}$, we can compute $f_i(p)$ for all terminals $p \in \partial P_i$ in time $O((n(P_{i-1}) + n(P_i)) \log n)$.*

Proof. We start with a structural observation. Let p be a terminal lying on the boundary of P_i. Let q be the point in P_{i-1} closest to p with $x(p) = x(q)$ (if it exists), and let q' be the point in P_{i-1} closest to p with $y(p) = y(q')$ (if it exists). Note that they are uniquely defined since P_{i-1} is ortho-convex. Let V_{i-1} be the set of vertices in P_{i-1}. We claim that $f_i(p) = \min\{f_{i-1}(r) + \mathrm{dist}(r, p) : r \in V_{i-1} \cup \{q, q'\}\}$.

Indeed, note that $q, q' \in \partial P_{i-1}$ due to step-disjointness. Since p is a portal and q, q' share a coordinate with p, both points are portals. Now consider a shortest tour that ends at p. By Corollary 11, we may assume that the tour visits either a vertex of P_{i-1} or q or q'. On the other hand, the concatenation of a tour realizing $f_{i-1}(r)$ and the tour $r \to p$ visits $P_1, \dots, P_i$ in order for any point $r \in V_{i-1} \cup \{q, q'\}$. Therefore, the claim holds.

As preprocessing, we compute $f_{i-1}(r)$ for all $r \in V_{i-1}$ using EVALUATE, and then construct the additively weighted Voronoi diagram of V_{i-1} where each point r has weight $f_{i-1}(r)$. This takes time $O(n(P_{i-1}) \log n)$.

After the preprocessing, we are ready to compute $f_i(p)$ for each terminal $p \in \partial P_i$. To this end, we query the Voronoi diagram to obtain $\min\{f_{i-1}(r) + \mathrm{dist}(r, p) : r \in V_{i-1}\}$. We compute the two terminals q, q' (as defined before) and $f_{i-1}(q), f_{i-1}(q')$ using EVALUATE. Finally, we can compute $f_i(p)$ using the claim above. Since the computation takes $O(\log n)$ time for each terminal by Lemma 24, and there are $O(n(P_{i-1}) + n(P_i))$ terminals on ∂P_i, the overall running time is $O((n(P_{i-1}) + n(P_i)) \log n)$. ◀

▶ **Lemma 26.** *Given dynamic mountain ranges f^T_{i-1} for all fragments $T \subset \partial P_{i-1}$, and $f_i(p)$ for all terminals $p \in \partial P_i$, we can compute f^S_i for any fragment $S \subset \partial P_i$ in time $O(\log n)$.*

Proof. We show how to do this for a horizontal fragment S of P_i; the vertical fragments can be handled symmetrically. Let terminals q, q' be the left and right ends of S. Let T be the horizontal fragment in P_{i-1} closest to S such that $x(S) = x(T)$. For any point $p \in S$, we claim that

$$f_i(p) = \min \begin{Bmatrix} f_{i-1}(\mathrm{proj}_T(p)) + |y(S) - y(T)|, \\ f_{i-1}(q) + x(p) - x(q), \\ f_{i-1}(q') + x(q') - x(p) \end{Bmatrix}.$$

To see this, observe that there are three possibilities of the last visited point $r \in P_{i-1}$ to reach p by Corollary 11. For the first case that $x(p) = x(r)$, since P_{i-1} is ortho-convex, all points in P_{i-1} with x-coordinate $x(p)$ must appear on the same side of S. Hence $r \in T$, or in other words $r = \mathrm{proj}_T(p)$. For the second case that $y(p) = y(r)$, since P_{i-1}, P_i are disjoint, $x(r) \notin [x(q), x(q')]$. In particular, the tour $r \to p$ goes through q or q'. For the third case that r is a vertex, $x(r) \notin (x(q), x(q'))$ by definition of fragments, thus the tour $r \to p$ goes through q or q'. Therefore, the claim holds.

To compute f^S_i, we first compute the closest horizontal fragment $T \subset \partial P_{i-1}$ with $x(S) = x(T)$. The first term in the claim can be obtained by $f^T_{i-1}.\textsc{shift}(|y(S) - y(T)|)$. The second term is a linear function in $x(p)$ with slope $+1$, namely $x(p) \mapsto x(p) + (f_{i-1}(q) - x(q))$. Similarly, the third term is a linear function in $x(q)$ with slope -1, namely $x(p) \mapsto -x(p) + (f_{i-1}(q') + x(q'))$. So the minimum can be computed by calling $\textsc{relax}(1, f_{i-1}(q) - x(q))$ and then $\textsc{relax}(-1, x(q') + f_{i-1}(q'))$. All the operations cost $O(\log n)$ time by Lemma 24. ◀

Chaining Lemmas 25 and 26, the description of iteration i is complete. Since each iteration takes $O((n(P_{i-1}) + n(P_i)) \log n)$ time, overall we need $O\left(\sum_{i=1}^{k} n(P_i) \log n\right) = O(n \log n)$ time. In the end, we return $\min_{p \in \mathrm{port}_k} f_k(p)$, which is correct by Corollary 11. Since $|\mathrm{port}_k| \leqslant 2n$ by ortho-convexity, we can find the minimum in $O(n \log n)$ time. This concludes the proof of Theorem 5.

References

1 Antonios Antoniadis, Mark de Berg, Sándor Kisfaludi-Bak, and Antonis Skarlatos. Computing smallest convex intersecting polygons. *Journal of Computational Geometry*, 16(1):167–202, 2025. doi:10.20382/jocg.v16i1a6.

2 Charles Joseph Argue, Anupam Gupta, Ziye Tang, and Guru Guruganesh. Chasing convex bodies with linear competitive ratio. *Journal of the ACM*, 68(5):32:1–32:10, 2021. doi:10.1145/3450349.

3 Nikhil Bansal, Martin Böhm, Marek Eliáš, Grigorios Koumoutsos, and Seeun William Umboh. Nested convex bodies are chaseable. *Algorithmica*, 82(6):1640–1653, 2020. doi:10.1007/s00453-019-00661-x.

4 Luis Barba, Jean Cardinal, John Iacono, Stefan Langerman, Aurélien Ooms, and Noam Solomon. Subquadratic algorithms for algebraic 3SUM. *Discrete & Computational Geometry*, 61(4):698–734, 2019. doi:10.1007/s00454-018-0040-y.

5 Sergei Bespamyatnikh. An $O(n \log n)$ algorithm for the zoo-keeper's problem. *Computational Geometry*, 24(2):63–74, 2003. doi:10.1016/S0925-7721(02)00092-5.

6 Karl Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 661–670. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.76.

7 Karl Bringmann. Fine-grained complexity theory (tutorial). In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*, volume 126 of *LIPIcs*, pages 4:1–4:7. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.STACS.2019.4.

8 Karl Bringmann, Sándor Kisfaludi-Bak, Marvin Künnemann, Dániel Marx, and André Nusser. Dynamic time warping under translation: Approximation guided by space-filling curves. *Journal of Computational Geometry*, 14(2):83–107, 2023. doi:10.20382/jocg.v14i2a6.

9 Karl Bringmann, Sándor Kisfaludi-Bak, Marvin Künnemann, André Nusser, and Zahra Parsaeian. Towards sub-quadratic diameter computation in geometric intersection graphs. In *38th International Symposium on Computational Geometry (SoCG 2022)*, volume 224 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.SoCG.2022.21.

10 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 79–97. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.15.

11 Karl Bringmann and André Nusser. Translating Hausdorff is hard: Fine-grained lower bounds for Hausdorff distance under translation. *Journal of Computational Geometry*, 13(2):30–50, 2022. doi:10.20382/jocg.v13i2a3.

12 Karl Bringmann, Frank Staals, Karol Węgrzycki, and Geert van Wordragen. Fine-grained complexity of Earth Mover's Distance under translation. In *40th International Symposium on Computational Geometry (SoCG 2024)*, volume 293 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.SoCG.2024.25.

13 Sébastien Bubeck, Yin Tat Lee, Yuanzhi Li, and Mark Sellke. Competitively chasing convex bodies. *SIAM Journal on Computing*, 52(2):STOC19–339–STOC19–353, 2023. doi:10.1137/20M1312332.

14 Timothy M. Chan, Hsien-Chih Chang, Jie Gao, Sándor Kisfaludi-Bak, Hung Le, and Da Wei Zheng. Truly subquadratic time algorithms for diameter and related problems in graphs of bounded VC-dimension, 2025. To appear in proceedings of FOCS 2025. doi:10.48550/arXiv.2510.16346.

15 Timothy M. Chan, Qizheng He, and Yuancheng Yu. On the fine-grained complexity of small-size geometric set cover and discrete k-center for small k. In *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 34:1–34:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ICALP.2023.34.

16 Panagiotis Charalampopoulos and Adam Karczmarz. Single-source shortest paths and strong connectivity in dynamic planar graphs. *Journal of Computer and System Sciences*, 124:97–111, 2022. doi:10.1016/J.JCSS.2021.09.008.

17 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2008. doi:10.1007/978-3-540-77974-2.

18 Mark de Berg and Marc van Kreveld. Rectilinear decompositions with low stabbing number. *Information processing letters*, 52(4):215–221, 1994. doi:10.1016/0020-0190(94)90129-5.

19 José Miguel Díaz-Báñez, Matias Korman, Pablo Pérez-Lantero, Alexander Pilz, Carlos Seara, and Rodrigo I. Silveira. New results on stabbing segments with a polygon. *Computational Geometry*, 48(1):14–29, 2015. doi:10.1016/j.comgeo.2014.06.002.

20 Moshe Dror, Alon Efrat, Anna Lubiw, and Joseph S. B. Mitchell. Touring a sequence of polygons. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing (STOC 2003)*, STOC '03, pages 473–482, 2003. doi:10.1145/780542.780612.

21 Jan Faigl, Vojtěch Vonásek, and Libor Přeučil. A multi-goal path planning for goal regions in the polygonal domain. In *Proceedings of the 5th European Conference on Mobile Robots (ECMR 2011)*, pages 171–176, 2011. URL: https://comrob.fel.cvut.cz/papers/ecmr11mtp.pdf.

22 Jan Faigl, Vojtěch Vonásek, and Libor Přeučil. Visiting convex regions in a polygonal map. *Robotics and Autonomous Systems*, 61(10):1070–1083, 2013. doi:10.1016/j.robot.2012.08.013.

23 Joel Friedman and Nathan Linial. On convex body chasing. *Discrete & Computational Geometry*, 9(3):293–321, 1993. doi:10.1007/BF02189324.

24 Anka Gajentaan and Mark H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Computational Geometry: Theory and Applications*, 5(3):165–185, 1995. doi:10.1016/0925-7721(95)00022-2.

25 Shu Ishida, Marc Rigter, and Nick Hawes. Robot path planning for multiple target regions. In *2019 European Conference on Mobile Robots (ECMR)*, pages 1–6. IEEE, 2019. doi:10.1109/ECMR.2019.8870971.

26 Rolf Klein. *Concrete and abstract Voronoi diagrams*. Springer, 1989. doi:10.1007/3-540-52055-4.

27 Miroslav Kulich, Jan Vidašič, and Jan Mikula. On the Travelling Salesman Problem with Neighborhoods in a Polygonal World. In José M. Cascalho, Mohammad Osman Tokhi, Manuel F. Silva, Armando Mendes, Khaled Goher, and Matthias Funk, editors, *Robotics in Natural Settings*, volume 530 of *Lecture Notes in Networks and Systems*, pages 334–345. Springer, Cham, 2023. doi:10.1007/978-3-031-15226-9_32.

28 Maarten Löffler and Marc van Kreveld. Largest and smallest convex hulls for imprecise points. *Algorithmica*, 56(2):235–269, 2010. doi:10.1007/s00453-008-9174-2.

29 Joseph S. B. Mitchell. Approximating watchman routes. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2013)*, pages 844–855. SIAM, 2013. doi:10.1137/1.9781611973105.60.

30 Joseph S. B. Mitchell and Micha Sharir. New results on shortest paths in three dimensions. In *Proceedings of the Twentieth Annual ACM Symposium on Computational Geometry*, SCG '04, pages 124–133, New York, NY, USA, 2004. Association for Computing Machinery. doi:10.1145/997817.997839.

31 Bengt J. Nilsson and Eli Packer. Approximation algorithms for the two-watchman route in a simple polygon. *Algorithmica*, 86(9):2845–2884, 2024. doi:10.1007/s00453-024-01245-0.

32 Justo Puerto and Carlos Valverde. The hampered travelling salesman problem with neighbourhoods. *Computers & Industrial Engineering*, 188:109889, 2024. doi:10.1016/j.cie.2024.109889.

33 Mark Sellke. Chasing convex bodies optimally. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2020)*, pages 1509–1518. SIAM, 2020. doi:10.1137/1.9781611975994.92.

34 Mark Sellke. Chasing convex bodies optimally. In *Geometric Aspects of Functional Analysis: Israel Seminar (GAFA) 2020-2022*, pages 313–335. Springer, 2023. doi:10.1007/978-3-031-26300-2_12.

35 Xuehou Tan and Tomio Hirata. Finding shortest safari routes in simple polygons. *Information Processing Letters*, 87(4):179–186, 2003. doi:10.1016/S0020-0190(03)00284-9.

36 Christophe Weibel and Linqiao Zhang. Minimum perimeter convex hull of imprecise points in convex regions. In *Proceedings of the Twenty-Seventh Annual Symposium on Computational Geometry (SoCG 2011)*, pages 293–294, New York, NY, USA, 2011. Association for Computing Machinery. doi:10.1145/1998196.1998243.