Towards Space Efficient Two-Point Shortest Path Queries in a Polygonal Domain

Sarita de Berg
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Tillmann Miltzow
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Frank Staals
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Abstract

We devise a data structure that can answer shortest path queries for two query points in a polygonal domain P on n vertices. For any $\varepsilon > 0$, the space complexity of the data structure is $O(n^{10+\varepsilon})$ and queries can be answered in $O(\log n)$ time. Alternatively, we can achieve a space complexity of $O(n^{9+\varepsilon})$ by relaxing the query time to $O(\log^2 n)$. This is the first improvement upon a conference paper by Chiang and Mitchell from 1999. They presented a data structure with $O(n^{11})$ space complexity and $O(\log n)$ query time. Our main result can be extended to include a space-time trade-off. Specifically, we devise data structures with $O(n^{9+\varepsilon}/\ell^{4+O(\varepsilon)})$ space complexity and $O(\ell \log^2 n)$ query time, for any integer $1 \leq \ell \leq n$.

Furthermore, we present improved data structures for the special case where we restrict one (or both) of the query points to lie on the boundary of P. When one of the query points is restricted to lie on the boundary, and the other query point is unrestricted, the space complexity becomes $O(n^{6+\varepsilon})$ and the query time $O(\log^2 n)$. When both query points are on the boundary, the space complexity is decreased further to $O(n^{4+\varepsilon})$ and the query time to $O(\log n)$, thereby improving an earlier result of Bae and Okamoto.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases data structure, polygonal domain, geodesic distance

Digital Object Identifier 10.4230/LIPIcs.SoCG.2024.17

Related Version *Full Version*: https://arxiv.org/abs/2303.00666

Funding *Tillmann Miltzow*: is generously supported by the Netherlands Organisation for Scientific Research (NWO) under project no. VI.Vidi.213.150.

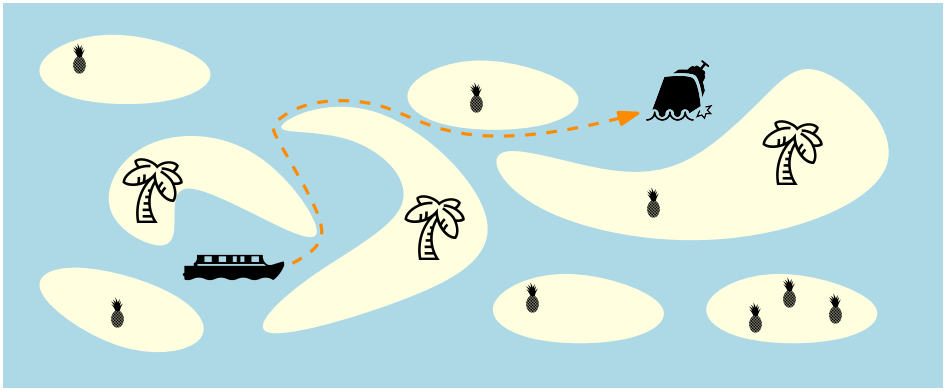

Figure 1 A tangible example of a two-point shortest path problem: finding the shortest path among islands for a boat to an emergency.

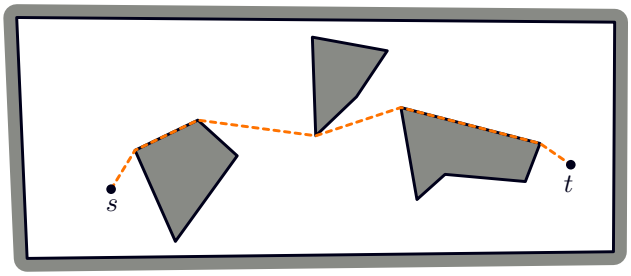

Figure 2 Given P and the query points s, t we want to compute the shortest path efficiently.

1 Introduction

In the Two-Point-Shortest-Path problem, we are given a polygonal domain P with n vertices, possibly containing holes, and we wish to store P so that given two query points $s, t \in P$ we can compute their *geodesic distance* $d(s, t)$, i.e. the length of a shortest path fully contained in P, efficiently. After obtaining this distance, the shortest path can generally be returned in $O(k)$ additional time, where k denotes the number of edges in the path. We therefore focus on efficiently querying the distance $d(s, t)$.

Tangible example. As an example of the relevance of the problem, consider a boat in the sea surrounded by a number of islands, see Figure 1. Finding the fastest route to an emergency, such as a sinking boat, corresponds to finding the shortest path among obstacles, i.e. in a polygonal domain. This is just one of many examples where finding the shortest path in a polygonal domain is a natural model of a real-life situation, which makes it an interesting problem to study.

Motivation. The main motivation to study the Two-Point-Shortest-Path problem is that it is a very natural problem. It is central in computational geometry, and forms a basis for many other problems. The problem was solved optimally for simple polygons (polygonal domains without holes) by Guibas and Hershberger [18], and turned out to be a key ingredient to solve many other problems in simple polygons. A few noteworthy examples are data structures for geodesic Voronoi diagrams [27], farthest-point Voronoi diagrams [34], k-nearest neighbor searching [1, 15], and more [16, 25]. In a polygonal domain, a two-point shortest path data structure is also the key subroutine in computing the geodesic diameter [5] or the geodesic center [32].

1.1 Related work

Chiang and Mitchell [12] announced a data structure for the Two-Point-Shortest-Path problem in polygonal domains at SODA 1999. They use $O(n^{11})$ space and achieve a query time of $O(\log n)$. They also present another data structure that uses "only" $O(n^{10} \log n)$ space, but $O(\log^2 n)$ query time. However, the paper refers to the full version for some of the details of these data structures, which never appeared. Since then, there have been no improvements on the two-point shortest path problem in its general form. Instead, related and restricted versions were considered. We briefly discuss the most relevant ones. Table 1 gives an overview of the related results.

Table 1 Overview of results on shortest paths in polygonal domains. All bounds are asymptotic. The parameter h represents the number of holes. The parameter q is the minimum of the number of vertices that s or t sees. The parameter f is the minimum number of faces needed to cover the vertices in a certain planar graph. The function $\lambda_m(n)$ is the maximum length of a Davenport-Schinzel sequence of n symbols of order m.

	Year	Paper	Space	Preprocessing	Query	Comments
two point shortest path	1989	[18]	n	n	$\log n$	simple polygon
	1999	[12]	n^{11}	n^{11}	$\log n$	
	1999	[12]	$n^{10}\log n$	$n^{10}\log n$	$\log^2 n$	
	1999	[12]	$n^{5+10\delta+\varepsilon}$	$n^{5+10\delta+\varepsilon}$	$n^{1-\delta}\log n$	$0 < \delta \leq 1$
	1999	[12]	$n + h^5$	?	$h \log n$	
	2001	[9]	n^2	$n^2 \log n$	$q \log n$	$q = O(n)$
	2008	[20]	n^2	$n^2 \log n$	$h \log n$	
	2012	[6]	$n^4\lambda_{66}(n)$	$n^4\lambda_{65}(n)\log n$	$\log n$	query points on ∂P
single source	1993	[26]	n	$n^{5/3}$	$\log n$	
	1999	[23]	n	$n \log n$	$\log n$	
	2021	[35]	n	$n \log n$	$\log n$	linear working space
approx-imation	1995	[8]	$\frac{n}{\varepsilon} + n \log n$	$o(f^{3/2}) + \frac{n \log n}{\varepsilon}$	$\frac{\log n}{\varepsilon} + \frac{1}{\varepsilon}$	Ratio $(6+\varepsilon)$
	2007	[31]	$\frac{n \log n}{\varepsilon}$	$\frac{n \log^3 n}{\varepsilon^2}$	$\frac{1}{\varepsilon^3} + \frac{\log n}{\varepsilon \log\log n}$	Ratio $(1+\varepsilon)$
L_1-metric	2000	[11]	$n^2 \log n$	$n^2 \log^2 n$	$\log^2 n$	
	2020	[33]	$n + \frac{h^2 \log^3 h}{\log\log h}$	$n + \frac{h^2 \log^4 h}{\log\log h}$	$\log n$	

As mentioned before, when the domain is restricted to a simple polygon, there exists an optimal linear size-data structure with $O(\log n)$ query time by Guibas and Hershberger [18].

When we consider the algorithmic question of finding the shortest path between two (fixed) points in a polygonal domain, the state-of-the-art algorithms build the so-called shortest path map from the source s [19, 22]. Hershberger and Suri presented such an $O(n)$-space data structure that can answer shortest path queries from a fixed point s in $O(\log n)$ time [23]. The construction takes $O(n \log n)$ time and space. This was recently improved by Wang [36] to run in optimal $O(n + h \log h)$ time and to use only $O(n)$ working space when a triangulation is given, where h denotes the number of holes in the domain.

By parameterizing the query time by the number of holes h, Guo, Maheshwari, and Sack [20] manage to build a data structure that uses $O(n^2)$ space and has query time $O(h \log n)$.

Bae and Okamoto [6] study the special case where both query points are restricted to lie on the boundary ∂P of the polygonal domain. They present a data structure of size $O(n^4\lambda_{66}(n)) \approx O(n^5)$ that can answer queries in $O(\log n)$ time. Here, $\lambda_m(n)$ denotes the maximum length of a Davenport–Schinzel sequence of order m on n symbols [30].

Two other variants that were considered are using approximation [8, 31], and using the L_1-norm [10, 11, 33]. Very recently, Hagedoorn and Polishchuk [21] considered two-point shortest path queries with respect to the link-distance, i.e. the number of edges in the path.

This seems to make the problem harder rather than easier: the space usage of the data structure is polynomial, and likely much larger than the geodesic distance data structures, but they do not provide an exact bound.

1.2 Results

Our main result is the first improvement in more than two decades that achieves optimal $O(\log n)$ query time.

▶ **Theorem 1** (Main Theorem). *For any constant $\varepsilon > 0$, we can build a data structure solving the Two-Point-Shortest-Path problem using $O(n^{10+\varepsilon})$ space and expected preprocessing time that has $O(\log n)$ query time. Alternatively, we can build a data structure using $O(n^{9+\varepsilon})$ space and expected preprocessing time that has $O(\log^2 n)$ query time.*

One of the main downsides of the two-point shortest path data structure is the large space usage. One strategy to mitigate the space usage is to allow for a larger query time. For instance, Chiang and Mitchell presented a myriad of different space-time trade-offs. One of them being $O(n^{5+10\delta+\varepsilon})$ space with $O(n^{1-\delta}\log n)$ query time for $0 < \delta \leq 1$. Our methods allow naturally for such a trade-off. We summarize our findings in the following theorem.

▶ **Theorem 2.** *For any constant $\varepsilon > 0$ and integer $1 \leq \ell \leq n$, we can build a data structure for the Two-Point-Shortest-Path problem using $O(n^{9+\varepsilon}/\ell^{4+O(\varepsilon)})$ space and expected preprocessing time that has $O(\ell \log^2 n)$ query time.*

For example, for $\ell = n^{3/4}/\log n$ we obtain an $O(n^{6+\varepsilon}\log^4 n)$-size data structure with query time $O(n^{3/4}\log n)$, which improves the $O(n^{7.5+\varepsilon})$-size data structure with similar query time of [12].

Another way to reduce the space usage is to restrict the problem setting. With our techniques it is natural to consider the setting where either one or both of the query points are restricted to lie on the boundary of the domain. In case we only restrict one of the query points to the boundary, we obtain the following result. Note that the other query point can lie anywhere in P.

▶ **Theorem 3.** *For any constant $\varepsilon > 0$, we can build a data structure for the Two-Point-Shortest-Path problem in $O(n^{6+\varepsilon})$ space and expected time that can answer queries for $s \in \partial P$ and $t \in P$ in $O(\log^2 n)$ time.*

When both query points are restricted to the boundary, we obtain the following result.

▶ **Theorem 4.** *For any constant $\varepsilon > 0$, we can build a data structure for the Two-Point-Shortest-Path problem in $O(n^{4+\varepsilon})$ space and time that can answer queries for $s \in \partial P$ and $t \in \partial P$ in $O(\log n)$ time.*

Note that the running time for this problem is deterministic. This improves the result by Bae and Okamoto [6], who provide an $O(n^4\lambda_{66}(n)) \approx O(n^5)$-sized structure for this problem.

To complement our positive algorithmic results, we also studied lower bounds. Unfortunately, we were only able to find a lower bound assuming that the Integer version of Hopcroft's problem takes $\Omega(n^2)$ time to compute. See the full version for a precise definition of Hopcroft's problem.

▶ **Theorem 5.** *Consider an data structure for the Two-Point-Shortest-Path problem with space complexity S and query time Q. Assuming the Integer Hopcroft lower bound, if Q is polylogarithmic then it holds that $S = \Omega(n^2)$.*

1.3 Discussion

In this section, we briefly highlight strengths and limitations of our results.

Applications. For many applications, our current data structure is not yet efficient enough to improve the state of the art. Yet, it is conceivable that further improvements to the two-point shortest path data structure will trigger a cascade of improvements for other problems in polygonal domains. One problem in which we do already improve the state of the art is in computing the geodesic diameter, that is, the largest possible (geodesic) distance between any two points in P. Bae et al. [5] show that with the two-point shortest path data structure of Chiang and Mitchell, the diameter can be computed in $O(n^{7.73+\varepsilon})$ time. By applying our improved data structure (with $\ell = n^{2/5}$), the running time can directly be improved to $O(n^{7.4+\varepsilon})$. Another candidate application is computing the geodesic center of P, i.e. a point that minimizes the maximum distance to any point in P. Wang [32] shows how to compute a center in $O(n^{11} \log n)$ time using two-point shortest path queries. Unfortunately, the two-point shortest path data structure is not the bottleneck in the running time, so our new data structure does not directly improve the result yet. Hence, more work is required here.

Challenges. Data structures often use divide-and-conquer strategies to efficiently answer queries. One of the main challenges in answering shortest path queries is that it is not clear how to employ such a divide and conquer strategy. We cannot easily partition the domain into independent subpolygons (the strategy used in simple polygons), as a shortest path may somewhat arbitrarily cross the partition boundary. Furthermore, even though it suffices to find a single vertex v on the shortest path from s to t (we can then use the shortest path map of v to answer a query), it is hard to structurally reduce the number of such candidate vertices. It is, for example, easily possible that both query points see a linear number of vertices of P, all of which produce a candidate shortest path of almost the same length. Hence, moving s or t slightly may result in switching to a completely different path.

We tackle these challenges by considering a set $\mathcal{T}$ of regions that come from the triangulated shortest path maps of the vertices of P. The number of regions in $\mathcal{T}$ is a measure of the remaining complexity of the problem. We can gradually reduce this number in a divide and conquer scheme using cuttings. However, initially we now have $O(n^2)$ regions as candidates to consider rather than just n vertices. Surprisingly, we show that we can actually combine this idea with a notion of *relevant pairs of regions*, of which there are only $O(n)$. This then allows us to use additional tools to keep the query time and space usage in check. It is this careful combination of these ideas that allows us to improve the space bound of Chiang and Mitchell [12].

Lower bounds. An important question is how much improvement of the space complexity is actually possible while retaining polylogarithmic query time. To the best of our knowledge there exist no non-trivial lower bounds on the space complexity of a two-point shortest path data structure. In the full version, we show a conditional quadratic lower bound. We conjecture that it may even be possible to obtain a super-quadratic lower bound. An indication for this is that the equivalence decomposition of P, which is the subdivision of P into cells such that the shortest path maps are topologically equivalent, has complexity $\Omega(n^4)$ [12]. However, this does not directly imply a lower bound on the space of any two-point shortest path data structure. We leave proving such a bound as an exciting open problem.

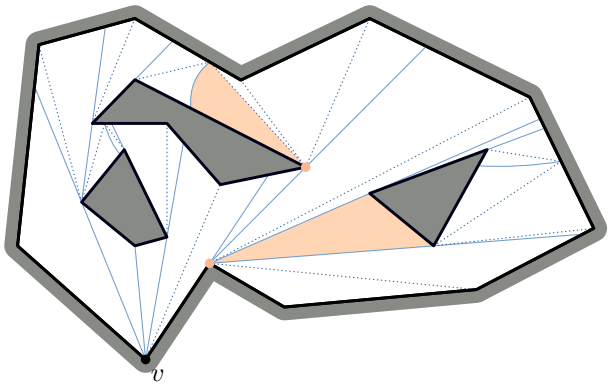

Figure 3 The augmented shortest path map of a vertex v. The shortest path map edges are solid, and the additional edges in the augmented shortest path map are dotted. Each region is bounded by three curves, of which at least two are line segments. Two regions and their apices are highlighted.

1.4 Organization

In Section 2, we give an overview of our main data structure. Then, in Section 3, we formally solve the subproblem where a subset of regions that contain s is provided. In Section 4, we tackle the two-point shortest path problem, and generalize this result to allow for a space-time trade-off. Lastly, in Section 5, we discuss our results for the restricted problem where either one or both of the query points must lie on the boundary of P. A full version of the paper is included in the appendix.

2 Overview of the approach

Direct visibility. As a first step, we build the visibility complex as described by Pocchiola and Vegter [28]. It allows us to query in $O(\log n)$ time if s and t can see each other. If so, the line segment connecting them is the shortest path. The visibility complex uses $O(n^2)$ space and can be built in $O(n^2)$ time. So, in the remainder, we assume that s and t cannot see each other, hence their shortest path will visit at least one vertex of P.

Augmented shortest path maps. In our approach, we build a data structure on the regions provided by the *augmented shortest path maps* of all vertices of P. The shortest path map of a point $p \in P$ is a partition of P into maximal regions, such that for every point in a region R the shortest path to p traverses the same sequence of vertices of P [23]. To obtain the augmented shortest path map $SPM(p)$, we refine the shortest path map by connecting each boundary vertex of a region R by a line segment with the apex v_R of that region, i.e. the first vertex on the shortest path from any point in R towards p. See Figure 3 for an example. All regions in $SPM(p)$ are "almost" triangles; they are bounded by three curves, two of which are line segments, and the remaining is either a line segment or a piece of a hyperbola. The (augmented) shortest path map has complexity $O(n)$ and can be constructed in $O(n \log n)$ time [23]. Let $\mathcal{T}$ be the multi-set of *all* augmented shortest path map regions of *all* the vertices of P. As there are n vertices in P, there are $O(n^2)$ regions in $\mathcal{T}$.

Because we are only interested in shortest paths that contain at least one vertex, the shortest path between two points $s, t \in P$ consists of an edge from s to some vertex v of P that is visible from s, a shortest path from v to a vertex u (possibly equal to v) that is visible from t, and an edge from u to t. For two regions $S, T \in \mathcal{T}$ with $s \in S$ and $t \in T$,

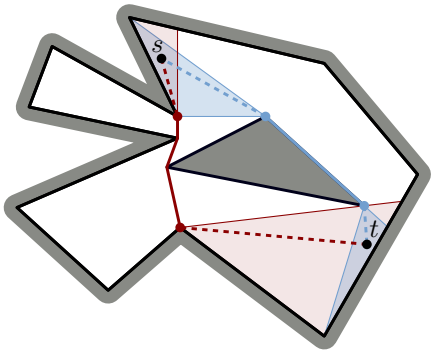

Figure 4 Two pairs of relevant regions in red and blue with the path whose length is $f_{ST}(s,t)$.

we define $f_{ST}(s,t) = ||sv_S|| + d(v_S, v_T) + ||v_T t||$. The distance $d(s,t)$ between s and t is realized by this function when $v_S = v$ and $v_T = u$. As for any pair S, T with $s \in S$ and $t \in T$ the function $f_{ST}(s,t)$ corresponds to the length of some path between s and t in P, we can obtain the shortest distance by taking the minimum over all of these functions. See Figure 4. In other words, if we denote by $\mathcal{T}_p$ all regions that contain a point $p \in P$, we have

$$d(s,t) = \min\{f_{ST}(s,t) : S \in \mathcal{T}_s, T \in \mathcal{T}_t\}.$$

Lower envelope. Given two multi-sets $\mathcal{A}, \mathcal{B} \subseteq \mathcal{T}$, we want to construct a data structure that we can efficiently query at any point (s,t) with $s \in \bigcap \mathcal{A}$ and $t \in \bigcap \mathcal{B}$ to find $\min\{f_{ST}(s,t) : S \in \mathcal{A}, T \in \mathcal{B}\}$. We refer to this as a Lower Envelope data structure. We can construct such a data structure of size $O(\min\{|\mathcal{A}|, |\mathcal{B}|, n\}^{6+\varepsilon})$ with $O(\log(\min\{|\mathcal{A}|, |\mathcal{B}|, n\}))$ query time, or of size $O(\min\{|\mathcal{A}|, |\mathcal{B}|, n\}^{5+\varepsilon})$ with $O(\log^2(\min\{|\mathcal{A}|, |\mathcal{B}|, n\}))$ query time, as follows.

The functions f_{ST} are four-variate algebraic functions of constant degree. Each such function gives rise to a surface in $\mathbb{R}^5$, which is the graph of the function f. Koltun [24] shows that the vertical decomposition of m such surfaces in $\mathbb{R}^5$ has complexity $O(m^{6+\varepsilon})$, and can be stored in a data structure of size $O(m^{6+\varepsilon})$ so that we can query the value of the lower envelope, and thus $d(s,t)$, in $O(\log m)$ time. More recently, Agarwal et al. [2] showed that a collection of m semialgebraic sets in $\mathbb{R}^5$ of constant complexity can be stored using $O(m^{5+\varepsilon})$ space such that vertical ray-shooting queries, and thus lower envelope queries, can be answered in $O(\log^2 m)$ time.

We limit the number of functions $f_{ST}(s,t)$ by using an observation of Chiang and Mitchell [12]. They note that we do not need to consider all pairs $S \in \mathcal{A}, T \in \mathcal{B}$, but only $\min\{|\mathcal{A}|, |\mathcal{B}|, n\}$ relevant pairs. Two regions form a *relevant* pair if they belong to the same augmented shortest path map $SPM(v)$ of some vertex v. (To be specific, if v is any vertex on the shortest path from s to t, then the minimum is achieved for S and T in the shortest path map of v.) We thus obtain a Lower Envelope data structure by constructing the data structure of [2] or [24] on these $\min\{|\mathcal{A}|, |\mathcal{B}|, n\}$ functions. This results in the following two lemmas, where $m = \min\{|\mathcal{A}|, |\mathcal{B}|, n\}$.

▶ **Lemma 6.** *For any constant $\varepsilon > 0$, we can construct a Lower Envelope data structure of size $O(m^{6+\varepsilon})$ in $O(m^{6+\varepsilon})$ expected time that can answer queries in $O(\log m)$ time.*

▶ **Lemma 7.** *For any constant $\varepsilon > 0$, we can construct a Lower Envelope data structure of size $O(m^{5+\varepsilon})$ in $O(m^{5+\varepsilon})$ expected time that can answer queries in $O(\log^2 m)$ time.*

Naively, to build a data structure that can answer shortest path queries for any pair of query points s, t, we would need to construct this data structure for all possible combinations of $\mathcal{T}_s$ and $\mathcal{T}_t$. The overlay of the n augmented shortest path maps has worst-case complexity $\Omega(n^4)$ [12], which implies that we would have to build $\Omega(n^8)$ of the Lower Envelope data structures. Indeed, this results in an $O(n^{14+\varepsilon})$-size data structure, and is one of the approaches Chiang and Mitchell consider [12]. Next, we describe how we use cuttings to reduce the number of Lower Envelope data structures we construct.

Cutting trees. Now, we explain how to determine $\mathcal{T}_s$ more efficiently using cuttings and cutting trees. Suppose we have a set $\mathcal{A}$ of N (not necessarily disjoint) triangles in the plane. A $1/r$-*cutting* Ξ of $\mathcal{A}$ is then a subdivision of the plane into constant complexity cells, for example triangles, such that each cell in Ξ is intersected by the boundaries of at most N/r triangles in $\mathcal{A}$ [7]. There can thus still be many triangles that fully contain a cell, but only a limited number whose boundary intersects a cell. The *size* of the cutting is the number of cells, and the *conflict list* of a cell is the set of triangles whose boundaries intersect the cell. In our case, the regions in $\mathcal{T}$ are *almost* triangles, called *Tarski cells* [4]. As we explain in the full version, we can always construct such a cutting with only $O(r^2)$ cells for these types of regions efficiently. The result is summarized in the following lemma.

▶ **Lemma 8.** *Let $\mathcal{T}$ be a set of N Tarski cells, and $r \in \{1, \ldots, N\}$ a parameter. A $1/r$-cutting Ξ of $\mathcal{T}$ of size $O(r^2)$, together with its conflict lists, can be computed in expected $O(Nr)$ time.*

Let Ξ be a $1/r$-cutting of $\mathcal{T}$. For $s \in \Delta \in \Xi$ the regions $R \in \mathcal{T}$ that fully contain Δ also contain s. To be able to find the remaining regions in $\mathcal{T}_s$, we recursively build cuttings on the N/r regions whose boundary intersects Δ. This gives us a so-called cutting tree. By choosing r appropriately, the cutting tree has constant height. The set $\mathcal{T}_s$ is then the disjoint union of all regions obtained in a root-to-leaf path in the cutting tree. Note that using a constant number of point location queries it is possible to find all of the vertices on this path.

The multi-level data structure. Our data structure, which we describe in detail in Sections 3 and 4, is essentially a multi-level cutting tree, as in [13]. See Figure 5 for an illustration. The first level is a cutting tree that is used to find the regions that contain s, as described before. For each cell Δ in a cutting Ξ, we construct another cutting tree to find the regions containing t. Let $\mathcal{A}$ be the set of regions fully containing Δ and $|\mathcal{A}| = k$. Then, the second-level cutting Ξ_Δ is built on the $O(kn)$ candidate regions that are in a relevant pair. See Figure 7. We process the regions intersected by a cell in Ξ_Δ recursively to obtain a cutting tree. Additionally, for each cell $\Delta' \in \Xi_\Delta$, we construct the Lower Envelope data structure on the sets $\mathcal{A}, \mathcal{B}$, where $\mathcal{B}$ is the set of regions that fully contain Δ'. This allows us to efficiently obtain $\min f_{ST}(s, t)$ for $S \in \mathcal{A}$ and $T \in \mathcal{B}$.

Queries. To query our data structure with two points s, t, we first locate the cell Δ_s containing s in the cutting Ξ at the root. We compute $\min f_{ST}(s, t)$ for all regions S that intersect Δ_s, but do not fully contain Δ_s, by recursively querying the child node corresponding to Δ_s. To compute $\min f_{ST}(s, t)$ for all S that fully contain Δ_s, we query its associated data structure. To this end, we locate the cell Δ_t containing t in Ξ_{Δ_s}, and use its lower envelope structure to compute $\min f_{ST}(s, t)$ over all S that fully contain Δ_s and all T that fully contain Δ_t. We recursively query the child corresponding to Δ_t to find $\min f_{ST}(s, t)$ over all T that intersect Δ_t.

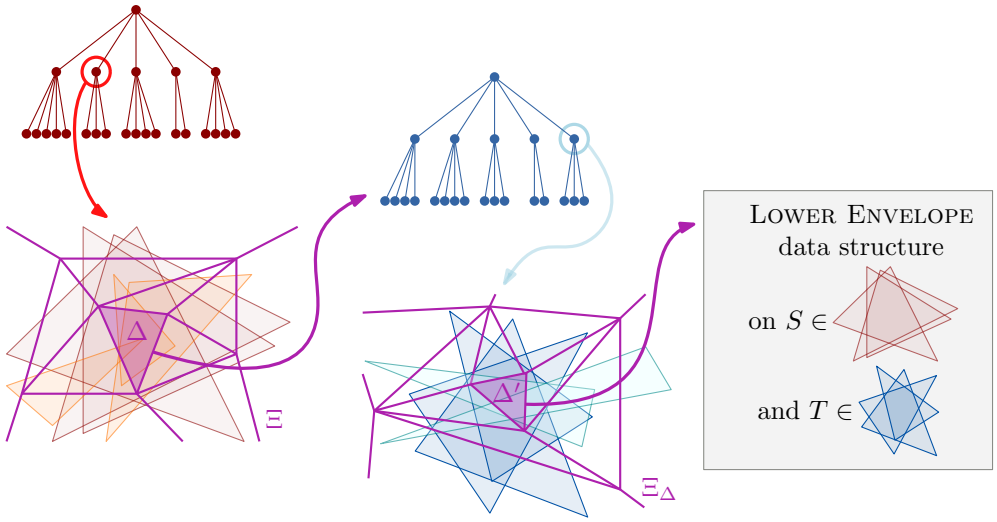

Figure 5 Overview of our data structure. The first level cutting tree (red) is built by recursively constructing a cutting Ξ on the (orange) regions that intersect a cell Δ (purple). For each cell Δ, we store a second level cutting tree (blue). For each cell Δ' in Ξ_Δ, we build a Lower Envelope data structure on all regions that fully contain Δ (dark red) and Δ' (dark blue).

Sketch of the analysis. By choosing r as n^δ for some constant $\delta = O(\varepsilon)$, we can achieve that each cutting tree has only constant height. The total query time is thus $O(\log n)$, when using the Lower Envelope data structure by Koltun [24]. Next, we sketch the analysis to bound the space usage of the first-level cutting tree, under the assumption that a second-level cutting tree, including the Lower Envelope data structures, uses $O(n^2 \min\{k, n\}^{6+\varepsilon})$ space (see Section 3).

To bound the space usage, we analyze the space used by the *large* levels, where the number of regions is greater than n, and the *small* levels of the tree separately, see Figure 6. There are only $O(n^2)$ large nodes in the tree. For these $\min\{k, n\} = n$, so each stores a data structure of size $O(n^{8+\varepsilon})$. For the small nodes, the size of the second-level data structures decreases in each step, as k becomes smaller than n. Therefore, the space of the root of a small subtree, which is $O(n^{8+\varepsilon})$, dominates the space of the other nodes in the subtree. As there are $O(n^2)$ small root nodes, the resulting space usage is $O(n^{10+\varepsilon})$.

A space-time trade-off. We can achieve a trade-off between the space usage and the query time by grouping the polygon vertices, see Section 4.1 for details. We group the vertices into ℓ groups, and construct our data structure on each set of $O(n^2/\ell)$ regions generated by a group. This results in a query time of $O(\ell \log^2 n)$ and a space usage of $O(n^{9+\varepsilon}/\ell^{4+O(\varepsilon)})$ when we apply the vertical ray-shooting Lower Envelope data structure of Lemma 7.

A data structure for s or t on the boundary. In Section 5, we show how to adapt our data structure to the case where one (or both) of the query points, say s, is restricted to lie on the boundary of the domain. The main idea is to use the same overall approach as before, but consider a different set of regions for s and t. The set of candidate regions for s now consists of intervals formed by the intersection of the *SPM* regions with the boundary of P. Instead of building a cutting tree on these regions, we build a segment tree, where nodes again store a cutting tree on the regions for t. As the functions $f_{ST}(s, t)$ are now only three-variate rather than four-variate, the resulting space usage is only $O(n^{6+\varepsilon})$.

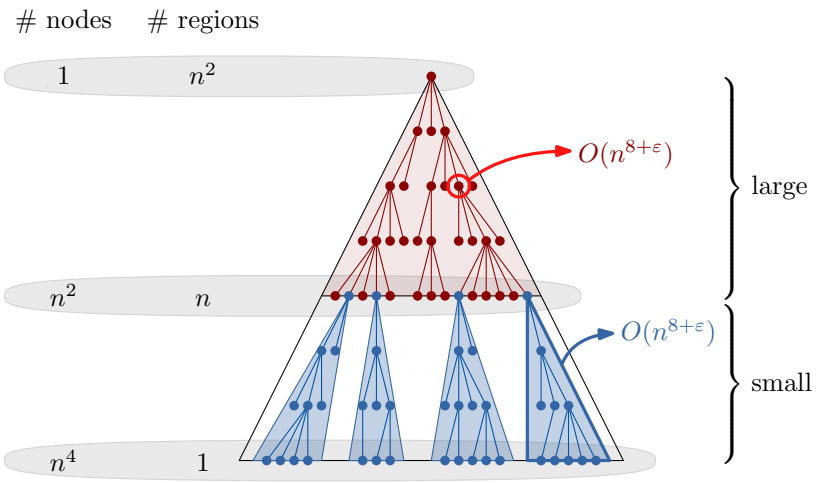

Figure 6 We analyze the large levels, built on $\geq n$ regions, and the small levels, built on $< n$ regions, separately. The total space usage is $O(n^{10+\varepsilon})$.

When both query point are restricted to the boundary, the set of candidate regions for both s and t consists of intervals along ∂P. In this case, a node of the segment tree stores another segment tree on the regions for t. Because the functions $f_{ST}(s,t)$ are only bivariate, the space usage is reduced to $O(n^{4+\varepsilon})$.

Conclusion

Improving the space bound. Both Lower Envelope data structures we use are actually more powerful than we require: one allows us to perform point location queries in the vertical decomposition of the *entire* arrangement, and the other allows us to perform vertical ray-shooting from *any* point in the arrangement. While we are only interested in lower envelope queries, i.e. vertical ray-shooting from a single plane. The (projected) lower envelope of m four-variate functions has a complexity of only $O(m^{4+\varepsilon})$ [29]. However, it is unclear if we can store this lower envelope in a data structure of size $O(m^{4+\varepsilon})$ while retaining the $O(\log m)$ query time. This would immediately reduce the space of our data structure to $O(n^{8+\varepsilon})$.

3 A data structure when $\mathcal{T}_s$ is given

In this section, we provide more details for the subproblem where we are given a set $\mathcal{A}$ of regions, and we want to build a data structure that can answer two-point shortest path queries for $s \in \bigcap \mathcal{A}$ and $t \in P$, see Figure 7. In particular, let $\mathcal{A}, \mathcal{B} \subseteq \mathcal{T}$ be two subsets of regions with $|\mathcal{A}| = k$, and $|\mathcal{B}| = M_0$. We develop a data structure that can efficiently compute $\mathcal{L}^{\mathcal{A}\times\mathcal{B}}(s,t) := \min\{f_{ST}(s,t) : S \in \mathcal{A}, T \in \mathcal{B} \text{ a relevant pair}\}$, provided that $s \in \bigcap \mathcal{A}$. This implies that $\mathcal{T}_s \supseteq \mathcal{A}$, and thus $k = O(n)$. Moreover, if $\mathcal{A} = \mathcal{T}_s$ and $\mathcal{B} = \mathcal{T}$ this thus allows us to compute $d(s,t)$ for any $t \in P$. As there are at most nk relevant pairs for t, we can assume that $M_0 \leq nk$. We can compute these regions, and store a bidirectional pointer between each relevant pair of regions in $\mathcal{A}$ and $\mathcal{B}$, in $O(n^2 \log n)$ time by sorting the regions on their respective shortest path map and performing a linear search. We formulate our result with respect to any Lower Envelope data structure that uses $S(m)$ space and expected preprocessing time on m functions, and has query time $Q(m)$, where $S(m)$ is of the form Cm^a for some constants $C > 0$ and $a \geq 3$, and $Q(m)$ is non-decreasing. In this section, we prove the following lemma.

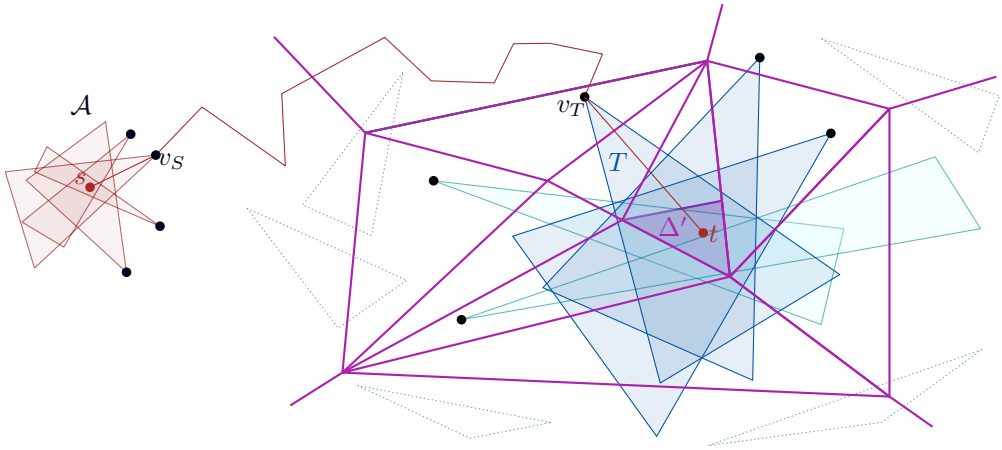

Figure 7 A sketch of the subproblem considered here, computing $\min_{S\in\mathcal{A},T\in\mathcal{T}} f_{ST}(s,t)$. We build a $1/r$-cutting Ξ_Δ (shown in purple) on the set of relevant regions in $\mathcal{T}$ (blue). The regions $\mathcal{T}_t \subseteq \mathcal{T}$ either fully contain the cell $\Delta' \in \Xi_\Delta$ of the cutting that contains t (dark blue), or their boundaries intersect Δ' (light blue).

▶ **Lemma 9.** *For any constant $\varepsilon > 0$, there is a data structure of size $O(n^{2+\varepsilon}S(k))$, so that for any query points s, t for which $s \in \bigcap \mathcal{A}$ we can compute $\mathcal{L}^{\mathcal{A}\times\mathcal{B}}(s,t)$ in $O(\log n + Q(k))$ time. Building the data structure takes $O(n^{2+\varepsilon}S(k))$ expected time.*

Note that if we are somehow given $\mathcal{A} = \mathcal{T}_s$, and have $\mathcal{B} = \mathcal{T}$ then $k = n$, and thus we get an $O(n^{2+\varepsilon}S(n))$-size data structure that can compute $d(s,t)$ in $O(\log n + Q(n))$ time.

Our data structure is essentially a cutting-tree [13] in which each node stores an associated Lower Envelope data structure. In detail, let $r \in \{2, \ldots, M_0\}$ be a parameter to be determined later. We build a $1/r$-cutting Ξ' of $\mathcal{B}$ using the algorithm from Lemma 8. Furthermore, we preprocess Ξ' for point location queries [17]. For each cell $\Delta \in \Xi'$, the subset $\mathcal{B}_\Delta$ of regions of $\mathcal{B}$ that contain Δ have an apex visible from any point within Δ. (Since all points in a region T see its apex v_T, i.e. the line segment to v_T does not intersect δP, and Δ is contained in T.) Hence, we store a Lower Envelope data structure on the pair of sets $(\mathcal{A}, \mathcal{B}_\Delta)$ for each cell Δ. We now recursively process the set $\mathcal{C}_\Delta$ of regions whose boundaries intersects Δ.

All that remains is to describe how to choose the parameter r that we use to build the cuttings. Let c be a constant so that the number of cells in a $1/r$-cutting on $\mathcal{B}$ has at most cr^2 cells. The idea is to pick $r = \max\{n^\delta, 2, 2c^{1/(a-2)}\}$, for some fixed $\delta \in (0,1)$ to be specified later.

Space usage. Let M be the number of regions from $\mathcal{B}$ in the current subproblem (initially $M = M_0$). Storing the cutting Ξ_Δ, and its point location structure takes $O(r^2)$ space [17]. Moreover, for each of the $O(r^2)$ cells, we store a Lower Envelope data structure of size $S(\min(k, M))$. There are only M/r regions whose boundary intersects a cell of the cutting on which we recurse. The space usage of the data structure is thus given by the following recurrence:

$$\mathcal{S}(M) = \begin{cases} cr^2\mathcal{S}(M/r) + O\left(r^2 \cdot S(\min\{k, M\})\right) & \text{if } M > 1 \\ O(1) & \text{if } M = 1. \end{cases}$$

After i levels of recursion, this gives $O(r^{2i}) = O(n^{2\delta i})$ subproblems of size at most $M_0/r^i \leq nk/n^{\delta i} = n^{1-\delta i}k$. It follows that at level $\frac{1}{\delta}$, we have $O(n^2)$ subproblems of size $O(k)$. Next, we analyze the space usage by the higher levels, where $i \leq \frac{1}{\delta}$, and lower levels, where $i > \frac{1}{\delta}$, separately. Note that for the higher levels, we have $\min\{k, M\} = k$, and for the lower levels $\min\{k, M\} = M$. There are only $\frac{1}{\delta} = O(1)$ higher levels.

In the higher levels of our data structure, the $O(r^{2i})$ associated Lower Envelope data structures at a level $i \leq \frac{1}{\delta}$ take $O(r^{2i} \cdot r^2 \cdot S(k)) = O(n^{2+\varepsilon}S(k))$ space, by setting $\delta = \varepsilon/2$. Since there are $O(1)$ higher levels, the total space usage of these levels is also $O(n^{2+\varepsilon}S(k))$.

In the lower levels of our data structure, we are left with $O(n^2)$ "small" subproblems at level $i = \frac{1}{\delta}$. For each such small subproblem, we have a $1/r$-cutting on $M \leq k$ regions that consists of at most cr^2 cells. Since $M \leq k$, the recurrence simplifies to

$$\mathcal{S}'(M) = \begin{cases} cr^2\mathcal{S}'(M/r) + O(r^2 S(M)) & \text{if } M > 1 \\ O(1) & \text{if } M = 1. \end{cases}$$

This recurrence solves to $O(r^2 S(M))$. A small subproblem thus uses $O(r^2 S(k))$ space, and therefore the lower levels use only $O(n^{2+\varepsilon}S(k))$ space in total.

Analyzing the preprocessing time. By Lemma 8 we can construct a $1/r$-cutting on $\mathcal{B}$, together with the conflict lists, in $O(rM)$ expected time. Since the cutting has size $O(r^2)$, we can also preprocess it in $O(r^2)$ time for point location queries [17]. Note that we can compute the set of relevant pairs in a subproblem in $O(\min\{k, M\})$ time, as we already computed pointers between each such pair. The expected time to build each associated structure is thus $S(\min\{k, M\})$. For the small subproblems, the $O(rM)$ term is dominated by the $O(r^2 S(M))$ time to construct the associated data structures, and thus we obtain the same recurrence as in the space analysis (albeit with different constants).

At each of the $O(1)$ higher levels, we spend $O(r^{2i} \cdot rM) = O(r^{2i+1}M_0/r^i) = O(n^{2\delta i+\delta}nk) = O(n^{2+\delta}k)$ expected time to construct the cutting, and $O(r^{2i}r^2 S(k)) = O(n^{2+2\delta}S(k))$ expected time to construct the associated data structures. Since the $O(n^{2+2\delta}S(k))$ term dominates, we thus obtain an expected preprocessing time of $O(n^{2+2\delta}S(k)) = O(n^{2+\varepsilon}S(k))$.

Querying. Let s, t be the query points. We use the point location structure on Ξ' to find the cell Δ that contains t, and query its associated Lower Envelope data structure to compute $\mathcal{L}^{\mathcal{A}\times\mathcal{B}_\Delta}(s, t)$. We recursively query the structure for the regions whose boundaries intersect Δ. When we have only $O(1)$ regions left in $\mathcal{B}_\Delta$, we compute $\mathcal{L}^{\mathcal{A}\times\mathcal{B}_\Delta}(s, t)$ by explicitly evaluating $f_{ST}(s, t)$ for all $O(1)$ pairs of relevant regions, and return the minimum found. A region that contains $t \in \Delta$ either contains a cell Δ, or intersects it. Moreover, all regions that contain Δ contain t. Hence, the sets $\mathcal{B}_\Delta$ over all cells Δ considered by the query together form a partition of the regions in $\mathcal{B}$ that contain t. Since we compute $\mathcal{L}^{\mathcal{A}\times\mathcal{B}_\Delta}(s, t)$ for each such set, our algorithm correctly computes $\mathcal{L}^{\mathcal{A}\times\mathcal{B}}(s, t)$, provided that $s \in \bigcap \mathcal{A}$.

Finding the cell Δ containing t takes $O(\log r)$ time, whereas querying the Lower Envelope data structure to compute $\mathcal{L}^{\mathcal{A}\times\mathcal{B}_\Delta}(s, t)$ takes $Q(\min\{k, M\})$ time. Hence, we spend $O(\log r + Q(k)) = O(\log n + Q(k))$ time at every level of the recursion. Observe that there are only $O(2/\delta) = O(1)$ levels in the recursion, as $M_0/r^{(2/\delta)} \leq n^2/n^{(\delta\cdot 2/\delta)} = 1$. It follows that the total query time is $O(\log n + Q(k))$ as claimed.

4 A two-point shortest path data structure

Let $\mathcal{T}$ be the set of all $N = O(n^2)$ shortest path map regions. To obtain a data structure that can be queried for $\mathcal{L}^{\mathcal{T}\times\mathcal{T}}(s,t) = d(s,t)$, we use a similar approach as in Section 3; i.e. we build a cutting tree that allows us to obtain $\mathcal{A}_s$ as $O(1)$-canonical subsets, and for each such set $\mathcal{A}$ we build a data structure from Lemma 9 such that we can compute $\mathcal{L}^{\mathcal{A}\times\mathcal{T}}(s,t)$.

As before, we build a $1/r$-cutting Ξ of $\mathcal{T}$, with $r = \max\{n^\delta, 2, 2c^{1/(a-2)}\}$ for some arbitrarily small $\delta > 0$, and preprocess it for point location queries [17]. For each cell $\Delta \in \Xi$, we consider the set $\mathcal{A}_\Delta$ of k regions fully containing Δ, and build the data structure from Lemma 9 on $(\mathcal{A}_\Delta, \mathcal{T})$ that can be efficiently queried for $\mathcal{L}^{\mathcal{A}_\Delta\times\mathcal{T}}(s,t)$ when $s \in \Delta$. We recursively process all regions from $\mathcal{T}$ whose boundaries intersect Δ (i.e. the at most n^2/r regions in the conflict list of Δ). Since the cutting consists of $O(r^2)$ cells, the space usage over all cells of Ξ is $O(r^2 n^{2+\varepsilon'} S(k))$. Let K be the number of regions from $\mathcal{T}$ in the current subproblem. Then, the space usage of the data structure follows the recurrence

$$\mathcal{S}_2(K) = \begin{cases} cr^2\mathcal{S}_2(K/r) + O\left(r^2 n^{2+\varepsilon'} S(\min\{K,n\})\right) & \text{if } K > 1 \\ O(1) & \text{if } K = 1. \end{cases}$$

Similar to Section 3, we can show this recurrence solves to $\mathcal{S}_2(K) = O(n^{4+\varepsilon}S(n))$ by analyzing the higher levels ($K > n$) and the lower levels ($K \leq n$) separately.

We can construct the set of regions $\mathcal{T}$ in $O(n^2 \log n)$ time, and store them using $O(n^2)$ space [23]. As mentioned before, constructing a data structure to test if s and t can see each other also takes $O(n^2 \log n)$ time $O(n^2)$ space. We thus obtain the following lemma.

▶ **Lemma 10.** *For any constant $\varepsilon > 0$, we can build a data structure using $O(n^{4+\varepsilon}S(n))$ space and expected preprocessing time that can answer two-point shortest path queries in $O(\log n + Q(n))$ time.*

By applying the lemma to the Lower Envelope data structures of Lemmas 6 and 7 we obtain our main result.

▶ **Theorem 1** (Main Theorem). *For any constant $\varepsilon > 0$, we can build a data structure solving the Two-Point-Shortest-Path problem using $O(n^{10+\varepsilon})$ space and expected preprocessing time that has $O(\log n)$ query time. Alternatively, we can build a data structure using $O(n^{9+\varepsilon})$ space and expected preprocessing time that has $O(\log^2 n)$ query time.*

▶ Remark. Note that it is also possible to avoid using a multi-level data structure and use only a single cutting tree for both s and t. In that case, we would build a Lower Envelope data structure for every pair of nodes of the cutting tree. However, we focus on the multi-level data structure here, as we do need this structure for the case where a query point is restricted to the boundary of P.

4.1 Space-time trade-off

We can achieve a trade-off between the space usage and the query time by grouping the polygon vertices. We group the vertices into ℓ groups $V_1, \ldots, V_\ell$ of size $O(n/\ell)$. We still compute the multi-set of regions $\mathcal{T}$ as before, but then partition the regions into ℓ multi-sets $\mathcal{T}_1, \ldots, \mathcal{T}_\ell$ where $\mathcal{T}_i$ contains all regions generated by sites in V_i. Note that each of these sets contains $O(n^2/\ell)$ regions. For each set $\mathcal{T}_i$, we build the data structure of Lemma 10. Each data structure is built on $O(n^2/\ell)$ regions, of which there are only $O(n/\ell)$ relevant pairs. In

the notation of Section 4, we have $N = O(n^2/\ell)$ regions and the space of the data structure of Lemma 9 is $O(n^{2+\varepsilon'}S(\min\{K, n/\ell\}))$. By using the same parameter choice for r as before, and considering a subproblem "small" whenever $K \leq n/\ell$, we obtain the following result.

▶ **Lemma 11.** *For any constant $\varepsilon > 0$ and integer $1 \leq \ell \leq n$, we can build a data structure in $O(\ell n^{4+\varepsilon}S(n/\ell))$ space and expected time that can answer two-point shortest path queries in $O(\ell(\log n + Q(n)))$ time.*

5 A data structure for s and/or t on the boundary

In this section, we sketch how to adapt our data structure to the case where one or both of the query points must lie on the boundary of the domain. In case only s is restricted to the boundary, but t can be anywhere in the interior of P, we obtain an $O(n^{6+\varepsilon})$ space data structure with $O(\log n)$ query time. In case both s and t are restricted to the boundary, the space usage decreases further to $O(n^{4+\varepsilon})$. This improves a result of Bae and Okamoto [6], who presented a data structure using roughly $O(n^{5+\varepsilon})$ space for this problem.

The main idea is to use the same overall approach as before: we subdivide the space into (possibly overlapping) regions, and construct a data structure that can report the regions stabbed by s as few canonical subsets. For each such subset, we build a data structure to find the regions stabbed by t and report them as canonical subsets, and for each of those canonical subsets, we store the lower envelope of the distance functions so that we can efficiently query $d(s, t)$. The two differences are that: (i) now the set of candidate regions for s and t might differ; for s this is now the set $\mathcal{I} = \{T \cap \partial P \mid T \in \mathcal{T}\}$ of intervals formed by the intersection of the *SPM* regions with the boundary of P; for t they are the set $\mathcal{T}$ or $\mathcal{I}$, depending on whether t is restricted to the boundary or not, and (ii) the functions $f_{ST}(s, t)$ are no longer four-variate. When only s is restricted to ∂P these functions are three-variate, and when both s and t are restricted to ∂P they are even bivariate.

Next, we describe the data structure where only s is restricted to lie on ∂P. Because the set of candidate regions for s is now the set $\mathcal{I}$ of intervals, rather than using cuttings to subdivide the regions as before, we use a segment tree $\mathfrak{T}$ with fanout $f = n^\delta$, for some $\delta \in (0, 1/2)$ [14]. Each leaf node μ in $\mathfrak{T}$ corresponds to some atomic interval J_μ. For each internal node ν we define J_ν as the union of the J_μ intervals of its children μ. Furthermore, for every node μ, let $\mathcal{I}_\mu$ denote the set of intervals from $\mathcal{I}$ that span J_μ, but do not span the interval J_ν of some ancestor ν of μ. Let k_μ denote the number of intervals in $\mathcal{I}_\mu$.

For each such a set $\mathcal{I}_\mu$, we build the data structure of Lemma 9. As our functions are now three-variate, there exists a Lower Envelope data structure that uses $O(m^{3+\varepsilon})$ space while supporting $O(\log^2 m)$ time queries [3], and thus the data structure from Lemma 9 uses $O(n^{2+\varepsilon}k_\mu^{3+\varepsilon})$ space. Each interval is stored at most $2f = 2n^\delta$ times at a level of the tree, and there are only $O(2/\delta) = O(1)$ levels, thus $\sum_{\mu\in\mathfrak{T}} k_\mu = O(n^{2+\delta})$. It then follows by setting ε' and δ appropriately that the space usage is

$$\sum_{\mu\in\mathfrak{T}} O\left(n^{2+\varepsilon'} \min\{k_\mu, n\}^{3+\varepsilon'}\right) = O\left(n^{2+\varepsilon'} \sum_{\mu\in\mathfrak{T}} n^{2+\varepsilon'} k_\mu\right) = O\left(n^{4+2\varepsilon'} \sum_{\mu\in\mathfrak{T}} k_\mu\right) = O(n^{6+\varepsilon}).$$

When both query points are restricted to the boundary, we also use a segment tree for the set of candidate regions for t. As the space of a Lower Envelope data structure is only $O(m^{2+\varepsilon})$ in this case, the space usage becomes $O(n^{4+\varepsilon})$.

References

1 Pankaj K. Agarwal, Lars Arge, and Frank Staals. Improved dynamic geodesic nearest neighbor searching in a simple polygon. In *Proceedings of the 34th International Symposium on Computational Geometry, SoCG*, volume 99 of *LIPIcs*, pages 4:1–4:14, 2018.

2 Pankaj K. Agarwal, Boris Aronov, Esther Ezra, and Joshua Zahl. Efficient algorithm for generalized polynomial partitioning and its applications. *SIAM J. Comput.*, 50(2):760–787, 2021. doi:10.1137/19M1268550.

3 Pankaj K. Agarwal, Boris Aronov, and Micha Sharir. Computing envelopes in four dimensions with applications. *SIAM J. Comput.*, 26(6):1714–1732, 1997. doi:10.1137/S0097539794265724.

4 Pankaj K. Agarwal and Jirí Matoušek. On range searching with semialgebraic sets. *Discret. Comput. Geom.*, 11:393–418, 1994. doi:10.1007/BF02574015.

5 Sang Won Bae, Matias Korman, and Yoshio Okamoto. The geodesic diameter of polygonal domains. *Discret. Comput. Geom.*, 50(2):306–329, 2013.

6 Sang Won Bae and Yoshio Okamoto. Querying two boundary points for shortest paths in a polygonal domain. *Comput. Geom.*, 45(7):284–293, 2012. doi:10.1016/J.COMGEO.2012.01.012.

7 Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discret. Comput. Geom.*, 9:145–158, 1993.

8 Danny Z. Chen. On the all-pairs Euclidean short path problem. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 292–301, 1995.

9 Danny Z. Chen, Ovidiu Daescu, and Kevin S. Klenk. On geometric path query problems. *Int. J. Comput. Geom. Appl.*, 11(06):617–645, 2001.

10 Danny Z. Chen, Rajasekhar Inkulu, and Haitao Wang. Two-point L_1 shortest path queries in the plane. *J. Comput. Geom.*, 7(1):473–519, 2016.

11 Danny Z. Chen, Kevin S. Klenk, and Hung-Yi T. Tu. Shortest path queries among weighted obstacles in the rectilinear plane. *SIAM Journal on Computing*, 29(4):1223–1246, 2000.

12 Yi-Jen Chiang and Joseph S. B. Mitchell. Two-point Euclidean shortest path queries in the plane. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 215–224, 1999. URL: http://dl.acm.org/citation.cfm?id=314500.314560.

13 Kenneth L. Clarkson. New applications of random sampling in computational geometry. *Discret. Comput. Geom.*, 2:195–222, 1987. doi:10.1007/BF02187879.

14 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications, 3rd Edition*. Springer, 2008.

15 Sarita de Berg and Frank Staals. Dynamic data structures for k-nearest neighbor queries. In *Proceedings of the 32nd International Symposium on Algorithms and Computation, ISAAC*, volume 212 of *LIPIcs*, pages 14:1–14:14, 2021.

16 Patrick Eades, Ivor van der Hoog, Maarten Löffler, and Frank Staals. Trajectory visibility. In *Proceedings of the 17th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT*, volume 162 of *LIPIcs*, pages 23:1–23:22, 2020. doi:10.4230/LIPICS.SWAT.2020.23.

17 Herbert Edelsbrunner, Leonidas J. Guibas, and Jorge Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.*, 15(2):317–340, 1986. doi:10.1137/0215023.

18 Leonidas J. Guibas and John Hershberger. Optimal shortest path queries in a simple polygon. *J. Comput. Syst. Sci.*, 39(2):126–152, 1989. doi:10.1016/0022-0000(89)90041-X.

19 Leonidas J. Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1):209–233, 1987.

20 Hua Guo, Anil Maheshwari, and Jörg-Rüdiger Sack. Shortest path queries in polygonal domains. In *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management, AAIM*, volume 5034 of *Lecture Notes in Computer Science*, pages 200–211. Springer, 2008. doi:10.1007/978-3-540-68880-8_20.

21 Mart Hagedoorn and Valentin Polishchuk. 2-point link distance queries in polygonal domains. In *Proceedings of the 39th European Workshop on Computational Geometry, EuroCG*, pages 229–234, 2023.

22 John Hershberger and Jack Snoeyink. Computing minimum length paths of a given homotopy class. *Computational Geometry*, 4(2):63–97, 1994.

23 John Hershberger and Subhash Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM J. Comput.*, 28(6):2215–2256, 1999. doi:10.1137/S0097539795289604.

24 Vladlen Koltun. Almost tight upper bounds for vertical decompositions in four dimensions. *J. ACM*, 51(5):699–730, 2004. doi:10.1145/1017460.1017461.

25 Matias Korman, André van Renssen, Marcel Roeloffzen, and Frank Staals. Kinetic geodesic Voronoi diagrams in a simple polygon. In *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming, ICALP*, volume 168 of *LIPIcs*, pages 75:1–75:17, 2020. doi:10.4230/LIPICS.ICALP.2020.75.

26 Joseph SB Mitchell. Shortest paths among obstacles in the plane. In *Proceedings of the 9th annual Symposium on Computational Geometry, SoCG*, pages 308–317, 1993.

27 Eunjin Oh. Optimal algorithm for geodesic nearest-point Voronoi diagrams in simple polygons. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 391–409. SIAM, 2019. doi:10.1137/1.9781611975482.25.

28 Michel Pocchiola and Gert Vegter. The visibility complex. *Int. J. Comput. Geom. Appl.*, 06(03):279–308, 1996.

29 Micha Sharir. Almost tight upper bounds for lower envelopes in higher dimensions. *Discret. Comput. Geom.*, 12:327–345, 1994. doi:10.1007/BF02574384.

30 Micha Sharir and Pankaj K. Agarwal. *Davenport-Schinzel sequences and their geometric applications*. Cambridge University Press, 1995.

31 Mikkel Thorup. Compact oracles for approximate distances around obstacles in the plane. In *Proceedings of the 15th Annual European Symposium, ESA*, volume 4698 of *Lecture Notes in Computer Science*, pages 383–394. Springer, 2007. doi:10.1007/978-3-540-75520-3_35.

32 Haitao Wang. On the geodesic centers of polygonal domains. *J. Comput. Geom.*, 9(1):131–190, 2018.

33 Haitao Wang. A divide-and-conquer algorithm for two-point L_1 shortest path queries in polygonal domains. *J. Comput. Geom.*, 11(1):235–282, 2020. doi:10.20382/JOCG.V11I1A10.

34 Haitao Wang. An optimal deterministic algorithm for geodesic farthest-point Voronoi diagrams in simple polygons. In *Proceedings of the 37th International Symposium on Computational Geometry, SoCG*, volume 189 of *LIPIcs*, pages 59:1–59:15, 2021.

35 Haitao Wang. Shortest paths among obstacles in the plane revisited. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 810–821. SIAM, 2021. doi:10.1137/1.9781611976465.51.

36 Haitao Wang. A new algorithm for Euclidean shortest paths in the plane. *J. ACM*, 70(2):11:1–11:62, 2023.