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

Sarita de Berg[1], Tillmann Miltzow[1], and Frank Staals[1]

1 **Department of Information and Computing Sciences, Utrecht University, The Netherlands**
s.deberg@uu.nl, t.miltzow@uu.nl, f.staals@uu.nl

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. This is the first improvement upon a conference paper by Chiang and Mitchell [8] from 1999. They present a data structure with $O(n^{11})$ space complexity. Furthermore, our main result can be extended to include a space-time trade-off. Specifically, we devise data structures with $O(n^{10+\varepsilon}/\ell^{5+O(\varepsilon)})$ space complexity and $O(\ell \log n)$ query time for any integer $1 \leq \ell \leq n$.

Related Version A full version of the paper is available at https://arxiv.org/abs/2303.00666.

1 Introduction

In the two-point shortest path problem, we are given a polygonal domain P with n vertices, 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, in $O(\log n)$ time.

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 [13], 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 [20], furthest point Voronoi diagram [25], k-th nearest neighbor search [1, 11], and more [12, 19]. In real life situations, the environment is often less restricted than a simple polygon. For example, consider a boat in the sea surrounded by a number of islands (Figure 2). 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.

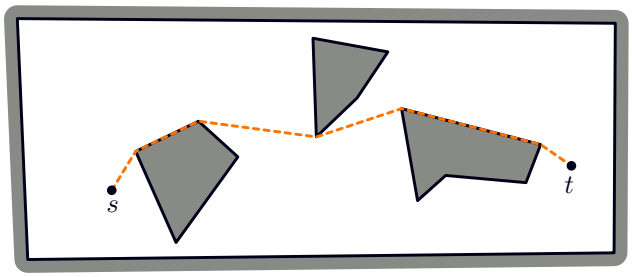

Figure 1 Given P and the query points s, t we want to compute the shortest path efficiently.

39th European Workshop on Computational Geometry, Barcelona, Spain, March 29–31, 2023.
This is an extended abstract of a presentation given at EuroCG'23. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

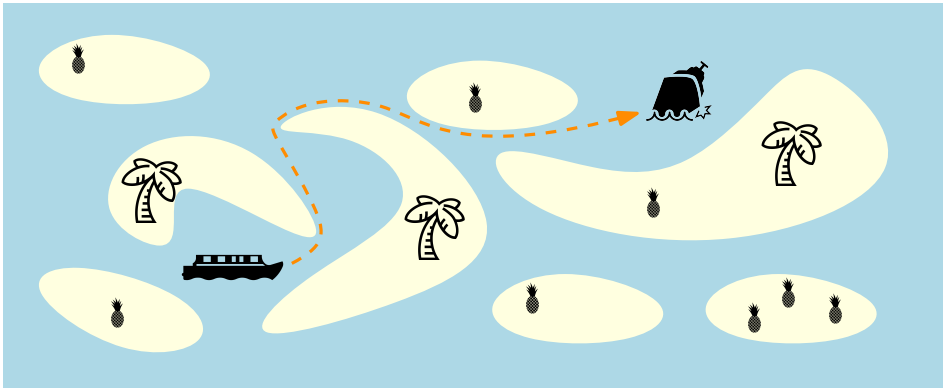

Figure 2 Finding the shortest path among islands for a boat to an emergency.

Related Work. Chiang and Mitchell [8] 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. 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.

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 [13].

By parameterizing the query time by the number of holes h, Guo, Maheshwari, and Sack [15] manage to build a data structure that uses $O(n^2)$ space and has query time $O(h \log n)$.

Bae and Okamoto [3] study the special case where both query points are restricted to lie on the boundary 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.

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 [14, 16]. 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 [13]. The construction takes $O(n \log n)$ time and space. This was recently improved by Wang [26] to run in $O(n + h \log h)$ time and to use only $O(n)$ working space.

Two other relaxations that were considered are approximation [5, 23], and using the L_1-norm [6, 7, 24].

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

▶ **Theorem 1.1** (Main Theorem). *Let P be a polygonal domain with n vertices. For any constant $\varepsilon > 0$, we can build a data structure in $O(n^{10+\varepsilon})$ space and expected time that can answer two-point shortest path queries in $O(\log n)$ time. The shortest path of k vertices can be returned in additional $O(k)$ time.*

One of the main downsides of the two-point shortest path data structure is the large space complexity. One strategy to mitigate the space complexity 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.

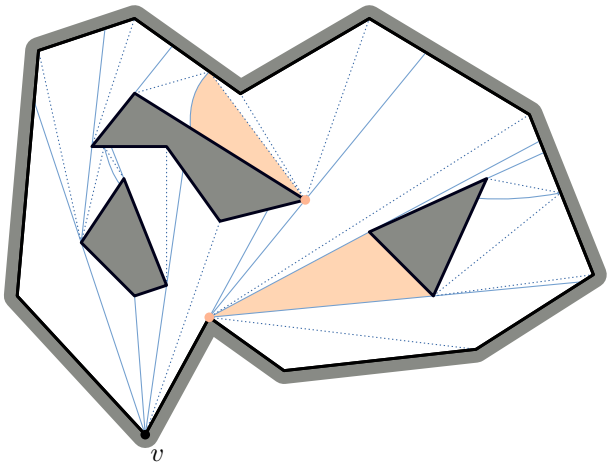

Figure 3 The augmented shortest path map of a vertex v. The shortest path map edges are solid, and the additional edges in the augmented shortest path map are dotted. Each region is bounded by three curves, of which at least two are line segments. Two regions and their apices are highlighted.

▶ **Theorem 1.2.** *Let P be a polygonal domain with n vertices. For any constant $\varepsilon > 0$ and integer $1 \leq \ell \leq n$, we can build a data structure in $O(n^{10+\varepsilon}/\ell^{5+O(\varepsilon)})$ space and expected time that can answer two-point shortest path queries in $O(\ell \log n)$ time.*

For example, for $\ell = n^{4/5}$ we obtain an $O(n^{6+\varepsilon})$ size data structure with query time $O(n^{4/5} \log n)$, which improves the $O(n^{7+\varepsilon})$ data structure with similar query time of [8].

Organization. In Section 2, we give an overview of our main data structure. A full version of the paper is available [10].

2 Global Approach

Direct Visibility. As a first step, we build the visibility complex as described by Pocchiola and Vegter [21]. 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 vertices of P [17]. To obtain the augmented shortest path map $SPM(p)$, we connect each boundary vertex of R with the apex v_R of the 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 $O(n)$ [17]. Let $\mathcal{T}$ be the multi-set of *all* augmented shortest path 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

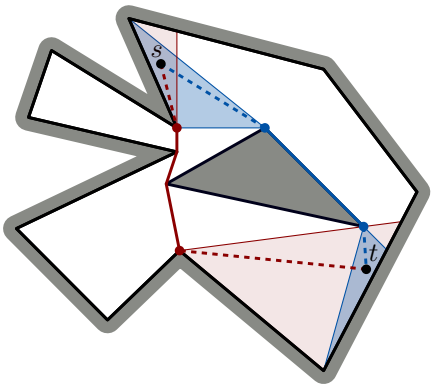

Figure 4 Two pairs of relevant regions in red and blue with the path whose length is $f_{ST}(s,t)$.

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 $t \in T$, we define $f_{ST}(s,t) = ||sv_S|| + d(v_S,v_T) + ||v_Tt||$. The distance $d(s,t)$ between s and t is realised 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 can construct a data structure of size $O(\min\{|\mathcal{A}|,|\mathcal{B}|,n\}^{6+\varepsilon})$ that we can query at any point (s,t) with $s \in \bigcap \mathcal{A}$ and $t \in \bigcap \mathcal{B}$ to find $\min\{f_{ST}(s,t) : S \in \mathcal{A}, T \in \mathcal{B}\}$ in $O(\log(\min\{|\mathcal{A}|,|\mathcal{B}|,n\}))$ time as follows. We refer to this as the Lower Envelope data structure.

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 [18] 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. We limit the number of functions $f_{ST}(s,t)$ by using an observation of Chiang and Mitchell [8]. 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 vertical decomposition of these $\min\{|\mathcal{A}|,|\mathcal{B}|,n\}$ functions.

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 $\Omega(n^4)$ [8], 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 [8]. Next, we describe how we use cuttings to reduce the number of Lower Envelope data structures we construct.

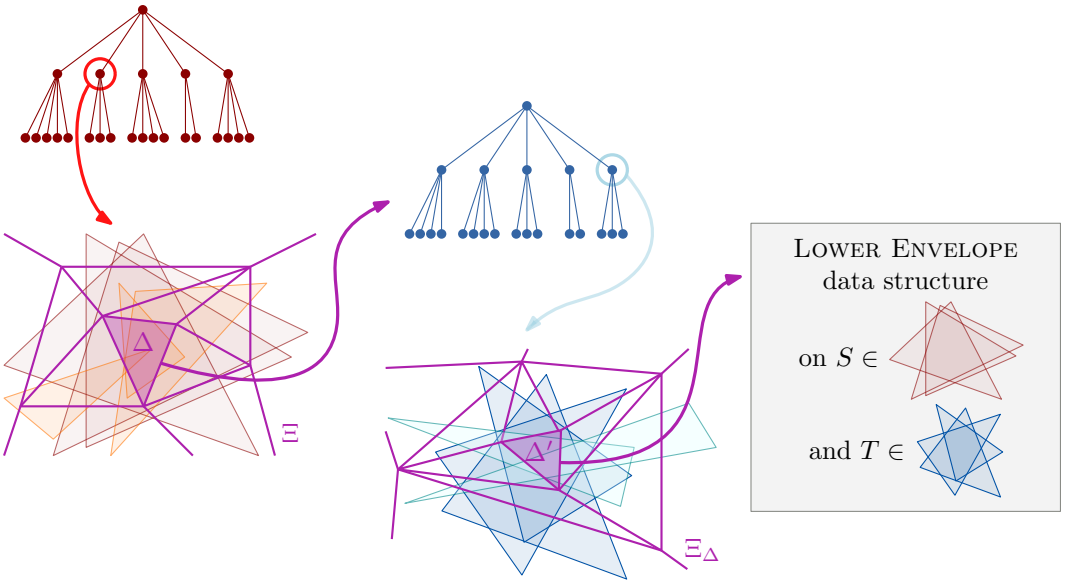

Figure 5 Overview of our data structure. The first level cutting tree (red) is built by recursively constructing a cutting Ξ on the (orange) regions that intersect a cell Δ (purple). For each cell Δ, we store a second level cutting tree (blue). For each cell Δ' in Ξ_Δ, we build a Lower Envelope data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

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 $\mathcal{A}$ [4]. There can thus still be many triangles that fully contain a cell, but only a limited number whose boundary intersects a cell. In our case, the regions in $\mathcal{T}$ are *almost* triangles, called *Tarski cells* [2]. As we explain in the appendix, we can always construct such a cutting with only $O(r^2)$ cells for these types of regions efficiently.

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. The set $\mathcal{T}_s$ is then the disjoint union of all regions obtained in a root to leaf path in the cutting tree.

The Multi-Level Data Structure. Our data structure is essentially a nested cutting tree, as in [9]. 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 $\Delta \in \Xi$ 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 relevant regions. See Figure 6. 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 obtain $\min f_{ST}(s,t)$ for $S \in \mathcal{A}$ and $T \in \mathcal{B}$ efficiently.

Queries. To query our data structure with two sites 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

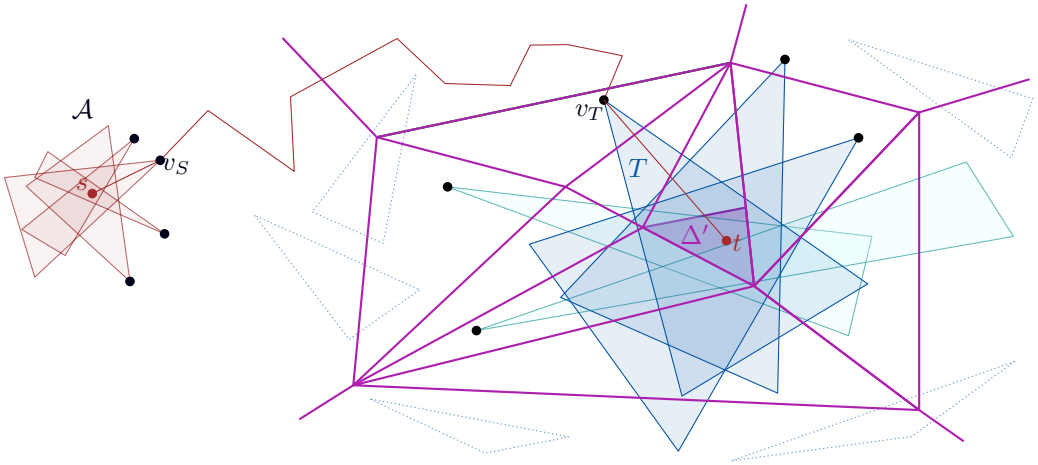

Figure 6 A sketch of the subproblem considered here, computing $\min_{S\in\mathcal{A},T\in\mathcal{T}} f_{ST}(s,t)$. We build a $1/r$-cutting Ξ_Δ (shown in purple) on the set of relevant regions in $\mathcal{T}$ (blue). The regions $\mathcal{T}_t \subseteq \mathcal{T}$ either fully contain the cell $\Delta' \in \Xi_\Delta$ of the cutting that contains t (dark blue), or their boundaries intersect Δ' (light blue).

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 regions T that intersect Δ_t.

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 $O(\log n)$. 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. The analysis for the second-level cutting tree is similar.

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 7. 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 $O(n^{10+\varepsilon})$.

3 Concluding remarks

The Lower Envelope data structure we use is actually more powerful than we require: it allows us to perform point location queries in the vertical decomposition of the entire arrangement, while we are only interested in lower envelope queries. The (projected) lower envelope of m four-variate functions has a complexity of only $O(m^{4+\varepsilon})$ [22]. 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. We are currently investigating if we can achieve such a bound using kinetic Voronoi diagrams. This would then immediately improve the space usage of our two-point shortest path data structure.

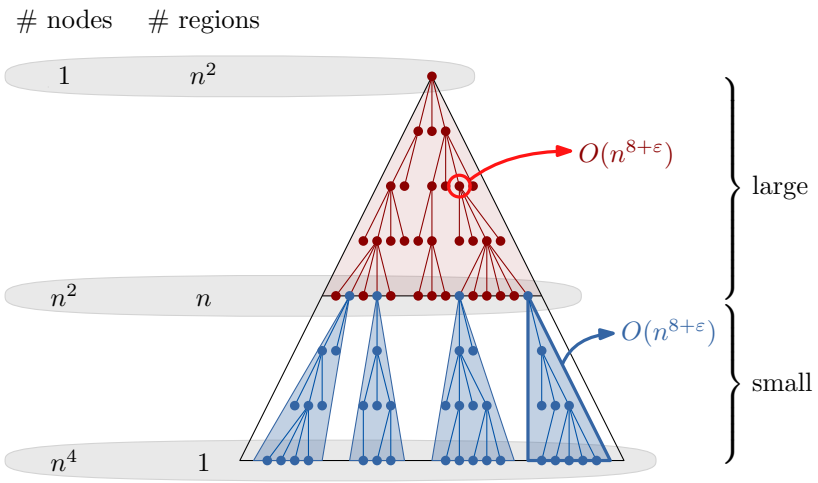

Figure 7 We analyze the large levels, built on $\geq n$ regions, and the small levels, built on $< n$ regions, separately. The total space usage is $O(n^{10+\varepsilon})$.

References

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

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

3 Sang Won Bae and Yoshio Okamoto. Querying two boundary points for shortest paths in a polygonal domain. *Comput. Geom.*, 45(7):284–293, 2012.

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

5 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.

6 Danny Z. Chen, Rajasekhar Inkulu, and Haitao Wang. Two-point L1 shortest path queries in the plane. *Journal of Computational Geometry*, 7(1):473–519, 2016.

7 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.

8 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.

9 Kenneth L. Clarkson. New applications of random sampling in computational geometry. *Discret. Comput. Geom.*, 2:195–222, 1987.

10 Sarita de Berg, Tillman Miltzow, and Frank Staals. Towards space efficient two-point shortest path queries in a polygonal domain. *CoRR*, abs/2303.00666, 2023.

11 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. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

12 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 2020, volume 162 of *LIPIcs*, pages 23:1–23:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

13 Leonidas J. Guibas and John Hershberger. Optimal shortest path queries in a simple polygon. *J. Comput. Syst. Sci.*, 39(2):126–152, 1989.

14 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.

15 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.

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

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

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

19 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. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

20 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.

21 Michel Pocchiola and Gert Vegter. The visibility complex. *Int. J. Comput. Geom. Appl.*, 06(03):279–308, 1996. doi:10.1142/S0218195996000204.

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

23 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.

24 Haitao Wang. A divide-and-conquer algorithm for two-point L1 shortest path queries in polygonal domains. *J. Comput. Geom.*, 11(1):235–282, 2020.

25 Haitao Wang. An optimal deterministic algorithm for geodesic farthest-point voronoi diagrams in simple polygons. In *37th International Symposium on Computational Geometry, SoCG 2021*, volume 189 of *LIPIcs*, pages 59:1–59:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

26 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.