To appear in *Proc. Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '99)*, January 17-19, 1999

Two-Point Euclidean Shortest Path Queries in the Plane

(Extended Abstract)

Yi-Jen Chiang* Joseph S. B. Mitchell†

Abstract

We consider the two-point query version of the fundamental geometric shortest path problem: Given a set h of polygonal obstacles in the plane, having a total of n vertices, build a data structure such that for any two query points s and t we can efficiently determine the length, $d(s,t)$, of an Euclidean shortest obstacle-avoiding path, $\pi(s,t)$, from s to t. Additionally, our data structure should allow one to *report* the path $\pi(s,t)$, in time proportional to its (combinatorial) size. We present various methods for solving this two-point query problem, including algorithms with $o(n)$, $O(\log n + h)$, $O(h \log n)$, $O(\log^2 n)$ or optimal $O(\log n)$ query times, using polynomial-space data structures, with various tradeoffs between space and query time. While several results have been known for *approximate* two-point Euclidean shortest path queries, it has been a well-publicized open problem to obtain sublinear query time for the exact version of the problem. Our methods also yield data structures for two-point shortest path queries on nonconvex polyhedral surfaces.

1 Introduction

Let P denote a *polygonal domain* in the plane, having n vertices and h holes; P is a closed, multiply-connected region whose boundary is a union of n line segments, forming $h+1$ closed (polygonal) cycles. (A *simple polygon*, then, is the special case of a polygonal domain with $h = 0$.) We will refer to P as "free space", and to the $h+1$ connected components, $\mathcal{O}$, of the complement of P (*h holes*, plus the *face at infinity*) as "obstacles." We let $\pi(s,t)$ denote an obstacle-avoiding (Euclidean) shortest path from $s \in P$ to $t \in P$; $d(s,t)$ will denote the length of $\pi(s,t)$.

In this paper, we consider the *two-point shortest-path query* problem, in which we are to construct a data structure that enables efficient processing of a query that specifies two points, s and t, and requests the length, $d(s,t)$, of a shortest path between them. The query may also request an actual instance of a shortest path. Since in general a path can be reported in additional time $O(k)$, where k is the number of edges in the output path, we concentrate on the complexity of performing a two-point distance query, to obtain $d(s,t)$.

While nearly-optimal results are known for the *single*-source query problem, prior to this work the two-point query problem has had no exact algorithm with sublinear query time; e.g., the recent survey [27] poses this open problem. Here, we provide several new results:

(1) An algorithm that uses $O(n^{5+10\delta+\epsilon})$ time and space to compute a data structure that supports $O(n^{1-\delta}\log n)$-time two-point shortest path queries, for any δ with $0 < \delta \leq 1$ and any fixed $\epsilon > 0$. In particular, this algorithm achieves slightly sublinear ($o(n)$) query time, with a data structure of size $O(n^{5+\epsilon})$. It also achieves optimal query time $O(\log n)$, with $O(n^{15+\epsilon})$ space.

(2) An algorithm that uses $O(n^{11})$ space and preprocessing to construct a data structure supporting optimal ($O(\log n)$-time) queries. Alternatively, the algorithm requires $O(n^{10}\log n)$ space and preprocessing in order to support $O(\log^2 n)$-time queries.

(3) Algorithms that are sensitive to the number, h, of obstacles. Since it may be that $h << n$, these methods offer important, potentially practical, alternatives to the high space complexities of the algorithms (1)-(2). Specifically, we obtain

  (a) query time $O(\log n + \min\{h_s, h_t\})$, using $O(n^5)$ space, where h_s (resp., h_t) is the number of "pivotal" (Section 5) obstacle vertices visible from s (resp., t), and h_s and h_t are bounded by $O(h)$; or

  (b) query time $O(h \log n)$, using $O(n + h^5)$ space.

*yjc@poly.edu. Department of Computer and Information Science, Polytechnic University, 6 MetroTech Center, Brooklyn, NY 11201. Most of this work was done while with the Department of Applied Mathematics and Statistics, SUNY at Stony Brook, and supported by NSF grant DMS-9312098 and a grant from Sandia National Labs.

†jsbm@ams.sunysb.edu. Department of Applied Mathematics and Statistics, State University of New York, Stony Brook, NY 11794-3600. Partially supported by NSF grant CCR-9504192, and by grants from Boeing Computer Services, Bridgeport Machines, Hughes Aircraft, Seagull Technology, and Sun Microsystems.

In particular, the bounds (b) match the optimal bounds ($O(\log n)$ query, $O(n)$ space) for simple polygons, in the case that the number h of holes is constant; these bounds interpolate between the optimal results known in the case of no holes, and the general results we obtain for any number ($O(n)$) of holes.

(4) Algorithms for two-point shortest path queries on nonconvex polyhedral surfaces, with space complexity a factor of n greater than in the planar problem.

These results are summarized in Table 1. We note that, while the space bounds for the optimal ($O(\log n)$) query time methods appear high, they should be compared with the best bounds known (see below) for the "simple" case of computing shortest paths on the surface of a *convex* polytope, where two-point queries are answered in time $O(\log n)$, using $O(n^{8+\epsilon})$ space and preprocessing time, for $\epsilon > 0$.

SPACE	QUERY TIME
$n^{5+\epsilon}$	$o(n)$
$n^{5+10\delta+\epsilon}$	$n^{1-\delta}\log n$, any $0 < \delta \leq 1$
$n^{10}\log n$	$\log^2 n$
n^{11}	$\log n$
n^5	$\log n + \min\{h_s, h_t\}$
$n + h^5$	$h \log n$

Our methods also develop and utilize structure of shortest path maps that may have independent interest. In particular, we devise a simple *parametric* point location method for shortest path maps, and make it dynamic (Section 4.2). We also introduce the notion of a *coarsened shortest path map* (Section 5), which we use in conjunction with the corridor structure of polygonal domains to obtain bounds that are sensitive to the number of holes. Further, we give the first polynomial bound ($O(n^{10})$) on the number of combinatorially distinct shortest path maps in a polygonal domain.

Related Work. There has been an abundance of work on geometric instances of the shortest path problem; we refer the reader to surveys in [19, 20, 27]. For the problem of computing Euclidean shortest paths in a polygonal domain, the best current results compute a single-source shortest path map (see the next section for definitions) in time and space $O(n \log n)$ ([14]), using the continuous Dijkstra paradigm ([18]), or compute shortest paths using visibility graph methods, in time $O(n + h^2 \log n)$ [15]. The best known lower bound is $\Omega(n + h \log h)$. After computing a shortest path map with respect to a given source point, shortest path queries to any destination can be answered in time $O(\log n)$.

In the case that P is a *simple* polygon ($h = 0$), Guibas and Hershberger [11] have shown how to preprocess P in time $O(n)$, into a data structure of size $O(n)$, to support two-point shortest-path queries in time $O(\log n)$.

Two-point Euclidean shortest-path queries in polygonal domains are considerably more challenging than the case of simple polygons. In a recent paper of Chen et al. [7], it has been shown that, using $O(n^2)$ space, one can achieve query time $O(K \log n)$, where $K = \min\{k_s, k_t\}$ is the smaller of the number of vertices visible from s and visible from t. However, this is worst-case $O(n \log n)$, which is no better than computing a shortest path map from scratch.

Given the difficulty of exact two-point queries, attention has focused on *approximate* two-point queries. As observed in [6], a method of Clarkson [8] can be used to construct a data structure of size $O(n^2)$, in $O(n^2 \log n)$ time, so that $(1+\epsilon)$-optimal queries can be answered in time $O(\log n)$, for any fixed $\epsilon > 0$. Chen [6] obtains nearly *linear*-space data structures for approximate shortest path queries, giving a $(6+\epsilon)$-approximation, using $O(n^{3/2}/\log^{1/2} n)$ time to build a data structure of size $O(n \log n)$, after which queries can be answered in time $O(\log n)$. These results have been improved recently by Arikati et al. [2], who give a family of results, based on planar spanners, with tradeoffs among the approximation factor and the preprocessing time, storage space, and query time.

For the problem of shortest paths on a polyhedral surface, some results are also known on the two-point query problem, at least for the case of *convex* polytopes. Agarwal et al [1] have also shown that two-point queries can be answered in time $O((\sqrt{n}/m^{1/4})\log n)$, with $O(n^6 m^{1+\delta})$ preprocessing time and storage, for any choice of $1 \leq m \leq n^2$, and $\delta > 0$. Har-Peled [12] obtains results for the *approximate* two-point query problem: He gives an $O(n)$-time algorithm to preprocess a convex polytope so that a two-point query can be answered in time $O((\log n)/\epsilon^{1.5} + 1/\epsilon^3)$, yielding the $(1+\epsilon)$-approximate shortest path distance, as well as a path having $O(1/\epsilon^{1.5})$ segments that avoids the interior of the input polytope.

2 Preliminaries

The input to our problem is a (multiply connected) polygonal domain, P, having h holes and a total of n vertices. We let V denote the set of vertices of P. We often refer to the complement of P (including the holes and the face at infinity) as *obstacles* and the vertices V as *obstacle vertices*. A query is specified by a pair of points, (s,t), with $s,t \in P$. We let $|\overline{pq}|$ denote the Euclidean length of the line segment $\overline{pq}$, and we let $d(p,q)$ denote the length of a shortest path, $\pi(p,q)$, joining $p \in P$ and $q \in P$.

The *visibility graph*, $VG(P)$, of P is the graph whose nodes correspond to vertices of P and whose edges link pairs of vertices that *see* one another. (Vertex u *sees* vertex v if the line segment $\overline{uv}$ lies within P.) $VG(P)$ can be computed in optimal output-sensitive time $O(e_{VG} + n \log n)$, where e_{VG} denotes the number of edges in the visibility graph [22]. It is a fundamental fact, based on local optimality, that any shortest path $\pi(s,t)$ must be a polygonal path that corresponds to a path in the visibility graph (after augmenting it with edges linking s and t to vertices visible from them).

For a point $z \in P$, the *visibility profile* of z, denoted $VP(z)$, is the locus of all points within P that are visible from z.

Given a *source point*, z, a *shortest path tree*, $SPT(z)$, is a spanning tree of z and the vertices of P such that the (unique) path in the tree between z and any vertex of P is a shortest path in P.

The *shortest path map*, $SPM(z)$, with respect to z, is a decomposition of P into regions (*cells*) according to the "combinatorial structure" of shortest paths from z to all other points. Refer to Fig. 1. Specifically, for all points p interior to a cell σ of $SPM(z)$, the sequence of obstacle vertices along $\pi(z,p)$ is fixed. In particular, the *last* obstacle vertex along $\pi(z,p)$ is the *root* of the cell σ containing p. We note that each cell is star-shaped with respect to its root, which lies on the boundary of the cell, and hence can be readily triangulated by connecting each vertex of the $SPM(z)$ to the root of the cells containing it. We define the *weight* of an obstacle vertex, v, to be $d(z,v)$. Typically, we will store with each vertex v both its weight, $d(z,v)$, and its *predecessor*, $r(v)$, which is the vertex (or point z) preceding v in a shortest path from z to v. Note that vertex v appears on the boundary of the (star-shaped) cell rooted at $r(v)$. The boundaries of cells consist of portions of obstacle edges, *extension segments* (extensions of visibility graph edges incident on the root), and *bisector curves*. The bisector curves are, in general, hyperbolic arcs that are the locus of points p that are equidistant (in the shortest path metric) from two distinct roots, u and v: points p satisfy $d(z,u) + |\overline{up}| = d(z,v) + |\overline{vp}|$. (Extension segments can be considered to be degenerate instances of bisector curves.)

Given point z, the shortest path map $SPM(z)$ is a (unique) planar subdivision of complexity $O(n)$, which can be constructed in $O(n \log n)$ time, using $O(n \log n)$ working storage [14].

If $SPM(z)$ is preprocessed for point location, then single-source shortest path queries can be answered efficiently by locating the query point t within the decomposition: If t lies in the cell rooted at r, the length of a shortest path to t is given by $d(z,t) = d(z,r) + |\overline{rt}|$. A shortest z-t path can then be output in time $O(k)$, where k is the number of vertices along the path, by simply following predecessor pointers back from r to z.

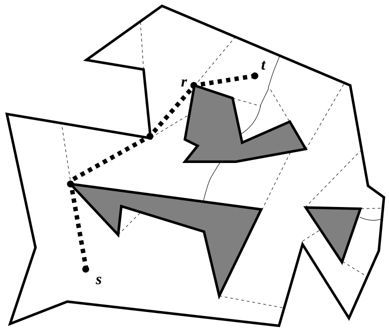

Figure 1: A shortest path map with respect to s.

We define $\mathcal{A}^{VP}$, the *VP-equivalence decomposition* of P, to be the subdivision of P into cells for which $VP(z)$ is combinatorially constant, as z varies within a cell. It is easy to see that $\mathcal{A}^{VP}$ is obtained by computing the arrangement of obstacle edges, together with the *extended* visibility graph edges. Thus, we see that $\mathcal{A}^{VP}$ has worst-case complexity $O(e_{VG}^2) = O(n^4)$; this bound is known to be tight.

Similarly, the *SPT-equivalence decomposition* of P, $\mathcal{A}^{SPT}$, is the subdivision of P into cells for which $SPT(z)$ is combinatorially constant. It is easy to see that $\mathcal{A}^{SPT}$ is obtained by overlaying the n shortest path maps, $SPM(v)$, for each vertex v of P. Since each $SPM(v)$ is of size $O(n)$, the overlay of these n subdivisions yields a decomposition of P of complexity $O(n^4)$; this bound is tight in the worst case. Note that $\mathcal{A}^{SPT}$ is a *refinement* of the decomposition $\mathcal{A}^{VP}$: each cell of $\mathcal{A}^{SPT}$ is a subcell of a cell of $\mathcal{A}^{VP}$, since having a constant SPT implies having a constant VP.

In Section 4, we will also define the "SPM-equivalence decomposition."

3 Method I: Mapping to Higher Dimensions

In this section, we show how the two-point query problem can be solved in optimal time ($O(\log n)$) by mapping it into a four-dimensional point location query. A related approach was used in [1] for the case of shortest path queries on a convex surface.

First, we note that a shortest path $\pi(s,t)$ from s to t either consists of the single segment $\overline{st}$ (if s sees t) or consists of a polygonal chain $(s, v_i, \ldots, v_j, t)$, whose bend points occur at obstacle vertices, where v_i is the obstacle vertex adjacent to s and v_j is the obstacle vertex adjacent to t. (Possibly, $v_i = v_j$.)

From now on, we assume that s is not visible from t; this is easily checked in time $O(\log n)$ (using space $O(n^2 \log n)$), using a two-point query structure $\mathcal{V}$.

Our goal now, for a given query point (x_s, y_s, x_t, y_t), is to minimize over all choices of first vertex (v_i) and last vertex (v_j). This is equivalent to evaluating at (x_s, y_s, x_t, y_t) the lower envelope function $f : \Re^4 \to \Re$, given by

$$f(x_s, y_s, x_t, y_t) = \min_{i \in I_s, j \in I_t} \left[|\overline{sv_i}| + d(v_i, v_j) + |\overline{v_j t}| \right],$$

where the minimum is taken over all choices ($i \in I_s$) of vertex v_i that sees s and all choices ($j \in I_t$) of vertex v_j that sees t. The index sets I_s and I_t depend on the coordinates (x_s, y_s, x_t, y_t); however, these index sets are *constant* for a given choice of $\mathcal{A}^{VP}$-cell σ_s containing s and $\mathcal{A}^{VP}$-cell σ_t containing t. Thus, for each of the $O(n^8)$ choices of the pair (σ_s, σ_t), we construct a data structure to compute the lower envelope of $f = \min_{i \in I_s, j \in I_t} f_{i,j}$ at a query point $(x_s, y_s, x_t, y_t) \in \Re^4$, where $f_{i,j}(x_s, y_s, x_t, y_t) = |\overline{sv_i}| + d(v_i, v_j) + |\overline{v_j t}|$. We have $O(n^2)$ functions, each of which is a surface in $\Re^5$, and we desire the lower envelope at a given point in $\Re^4$. This can be done in query time $O(\log n)$ using a data structure of size $O((n^2)^{2 \cdot 5 - 3 + \epsilon}) = O(n^{14+\epsilon})$, using known results on higher-dimensional point location, based on decompositions of arrangements of real algebraic surfaces (e.g., see Section 8.3, [25]). Since this structure is built for each of the $O(n^8)$ choices of (σ_s, σ_t), we have overall $O(n^{22+\epsilon})$ space, for $O(\log n)$ query time.

Using an alternative mapping into a lower envelope problem, we can improve the space complexity, as follows. Since we are assuming that s does not see t (as this is the trivial case), we know that there is at least one obstacle vertex v_i on $\pi(s,t)$ and that the length of $\pi(s,t)$ is given by $d(s, v_i) + d(v_i, t)$, for *any* choice of v_i on an optimal path. (v_i need not be visible to s or to t, since we are using geodesic distances $d(s, v_i)$ and $d(v_i, t)$.) Thus, $d(s,t) = \min_i d(s, v_i) + d(v_i, t)$, where the minimum is taken over all vertices $v_i \in V$. We seek, at the given query point, the lower envelope of the n functions of the form

$$g_i(x_s, y_s, x_t, y_t) = d(s, v_i) + d(v_i, t).$$

Now, these functions have a special structure: each is a function of four variables that separates into a sum of two functions, each of two variables. Further, each of the two-variable functions ($d(s, v_i)$ and $d(v_i, t)$) is encoded in the shortest path map $SPM(v_i)$ rooted at v_i, which itself describes a surface in three dimensions. In particular, $g_i(x_s,\, y_s,\, x_t,\, y_t)$ can be written explicitly if we know the root (u_i) of the cell of $SPM(v_i)$ that contains s and the root (w_i) of the cell of $SPM(v_i)$ that contains t:

$$\begin{aligned} g_i(x_s, y_s, x_t, y_t) \quad &= \sqrt{(x_s - x_{u_i})^2 + (y_s - y_{u_i})^2} + d(u_i, v_i) \\ &\quad + d(v_i, w_i) + \sqrt{(x_t - x_{w_i})^2 + (y_t - y_{w_i})^2}. \end{aligned}$$

The cell σ_s (resp., σ_t) of $\mathcal{A}^{SPT}$ that contains s (resp., t) gives us the identity of the root u_i (resp., w_i), for every choice of v_i. Thus, we construct $\mathcal{A}^{SPT}$ and store with each cell the list of roots corresponding to it, for each choice of v_i. This requires $O(n^5)$ space. Also, for each of the $O(n^8)$ choices of the pair (σ_s, σ_t) of cells, we construct a data structure, of size $O(n^{2 \cdot 5 - 3 + \epsilon}) = O(n^{7+\epsilon})$, to support $O(\log n)$ lower envelope queries on the n functions g_i. The overall space bound is $O(n^{15+\epsilon})$.

By trading off space for query time, we are able to obtain a substantial reduction in the size of our data structures, while allowing the query time to increase (though remain sublinear). The full paper describes in detail how we do this, utilizing a partition of the vertex set V into $m = n^{1-\delta}$ sets ($C_1, \ldots, C_m$) each of size n^δ for a parameter $\delta \in (0, 1]$.

THEOREM 3.1. *Using $O(n^{5+10\delta+\epsilon})$ time and space, one can compute a data structure that supports $O(n^{1-\delta} \log n)$-time two-point shortest path queries in a polygonal domain in the plane. Here, δ is any fixed parameter satisfying $0 < \delta \le 1$ and $\epsilon > 0$ is any fixed positive number. In particular, $O(\log n)$-time queries can be performed using $O(n^{15+\epsilon})$ space, and sublinear ($o(n)$) queries can be performed using $O(n^{5+\epsilon})$ space.*

4 Method II: SPM-Equivalence Decompositions

In this section, we give a different method for query processing, which yields improved space bounds on the data structure to support logarithmic (or polylogarithmic) query bounds.

We define the *SPM-equivalence decomposition*, $\mathcal{A}^{SPM}$, of P to be the subdivision of P into cells such that for all points z in the same cell σ of $\mathcal{A}$, the shortest path maps $SPM(z)$ are topologically equivalent. (We say that two shortest path maps are *topologically equivalent* if their underlying plane graphs are isomorphic.) For simplicity of notation, we will use the term *equivalence decomposition* and write simple $\mathcal{A}$, instead of $\mathcal{A}^{SPM}$, in this section. Note that $\mathcal{A}$ is a *refinement* of the decomposition $\mathcal{A}^{SPT}$, since having distinct shortest path trees implies having topologically distinct shortest path maps.

Our method consists of the following preprocessing steps:

1. Construct the equivalence decomposition, $\mathcal{A}$, and an associated point-location data structure.

2. For each cell σ of $\mathcal{A}$, compute the shortest path map $SPM(\sigma)$, whose underlying plane graph is the plane graph of $SPM(z)$ for any point $z \in \sigma$, and whose geometric realization is *parameterized*; i.e., each of the geometric constituents (vertices and edges) is expressed algebraically as a function of the coordinates of $z \in \sigma$, thereby allowing the actual object to be computed in time $O(1)$ for a given point z.

3. For each cell σ, build a point-location data structure $\mathcal{D}(\sigma)$ for $SPM(\sigma)$. This data structure serves for locating a query point q in $SPM(z)$ for any $z \in \sigma$ and is again *parameterized*; i.e., each partitioning object stored and used for comparison during a search (in processing a query) is represented by an equation such that its exact position can be computed in $O(1)$ time, given a source point $z \in \sigma$.

In order to process a two-point shortest path query, for points s and t, we first locate the cell σ of $\mathcal{A}$ containing s; this point location in the (static) subdivision $\mathcal{A}$ can be done using a data structure (of size $|\mathcal{A}|$), by any of the existing optimal techniques [9, 16, 24]. Then, to determine the shortest path length $d(s,t)$, we perform a point location query for t in $SPM(s)$. This, however, requires that we develop a new point location data structure that supports *parametric* point location in $SPM(s)$. For this, we exploit the structure of the map $SPM(s)$, as we will discuss below. First, we analyze the structure and complexity of the equivalence decomposition, $\mathcal{A}$.

4.1 Equivalence Decomposition $\mathcal{A}$. Simple examples show that $\mathcal{A}^{VP}$ can have complexity $\Omega(n^4)$; since $\mathcal{A}^{SPT}$ is a refinement of $\mathcal{A}^{VP}$, and $\mathcal{A} = \mathcal{A}^{SPM}$ is a refinement of $\mathcal{A}^{SPT}$, this also shows that $\Omega(n^4)$ is a lower bound on the complexity of $\mathcal{A}$. In the case of $\mathcal{A}^{VP}$ and $\mathcal{A}^{SPT}$, there is a matching $O(n^4)$ upper bound (see remark in Section 2). We do not yet know a tight upper bound on $|\mathcal{A}|$; the following analysis of the structure of $\mathcal{A}$ gives the best upper bound we know.[1]

The decomposition $\mathcal{A}$ arises from an arrangement of two types of curves: "bisector curves" and "topology curves." The *bisector curves* are the $O(n^2)$ hyperbolic arcs (possibly degenerated to straight lines) of the n shortest path maps, $SPM(v)$, having sources at each vertex v. A *topology curve* c is a curve such that, as z crosses the curve, the topology of the underlying plane graph of $SPM(z)$ changes, as an edge contracts or expands; i.e., moving z across c is an event of a topological change in $SPM(z)$.

[1]We thank M. Sharir for discussions on this analysis.

Events that cause an edge contraction/expansion in $SPM(z)$ can be classified according to the type of the SPM-edge and the type of its endpoints. SPM-edges are either bisector (B) edges or obstacle (O) edges. The endpoints of a bisector edge can be classified as "BB" or "BO," depending on what type of edge is incident at the endpoint: an endpoint is "BB" if only bisector edges are incident on it, and it is "BO" otherwise. Similarly, the endpoints of an obstacle edge can be classified as "OO" (meaning am obstacle vertex) or "BO." Bisector edges are of three possible types, then: (BB,BB), (BB,BO), and (BO,BO). One can show, however, that a bisector edge of type (BO,BO) can never contract to zero length as z varies. (It would have to contract to an obstacle vertex, "trapping" the region to one side of the bisector, allowing no path back to z.) Obstacle edges in $SPM(z)$ are also of three possible types: (OO,OO), (OO,BO), and (BO,BO). An edge of type (OO,OO) does not contract to a point.

In the nondegenerate situation, every node of the graph $SPM(z)$ is of degree three. When z moves to a point, as it crosses a topology curve c, that causes one or more SPM-edges to contract, higher degree nodes arise. We can classify the possible events (and corresponding topology curve c) according to the type of edge contraction, as follows:

(1) *(BB,BB) bisector edge contracts: Fig. 2(a).* At the point q of contraction, four bisector curves of $SPM(z)$ meet at q; q then is a degree-4 node in $SPM(z)$, and there exist four distinct shortest paths from z to q.

(2) *(BB,BO) bisector edge contracts: Fig. 2(b).* At the point q of contraction, two bisector curves of $SPM(z)$ meet with an obstacle edge at point q; q then is a degree-4 node in $SPM(z)$ and there exist three distinct shortest paths from z to q.

(2)' *(BO,BO) obstacle edge contracts: Fig. 2(b).* This event is symmetric with the event of Type (2), since the contraction of a (BB,BO) bisector edge is followed by the expansion of a (BO,BO) obstacle edge, and vice versa.

(3) *(OO,BO) obstacle edge contracts: Fig. 2(c).* At the point $q = v$ of contraction, a bisector curve of $SPM(z)$ meets the obstacle vertex v; v then is a degree-3 node in $SPM(z)$ and there exist two distinct shortest paths from z to v. In other words, z lies on a bisector in $SPM(v)$.

Since a type (3) curve c is simply a bisector curve in $SPM(v)$, we concentrate on type (1) and type (2) topology curves.

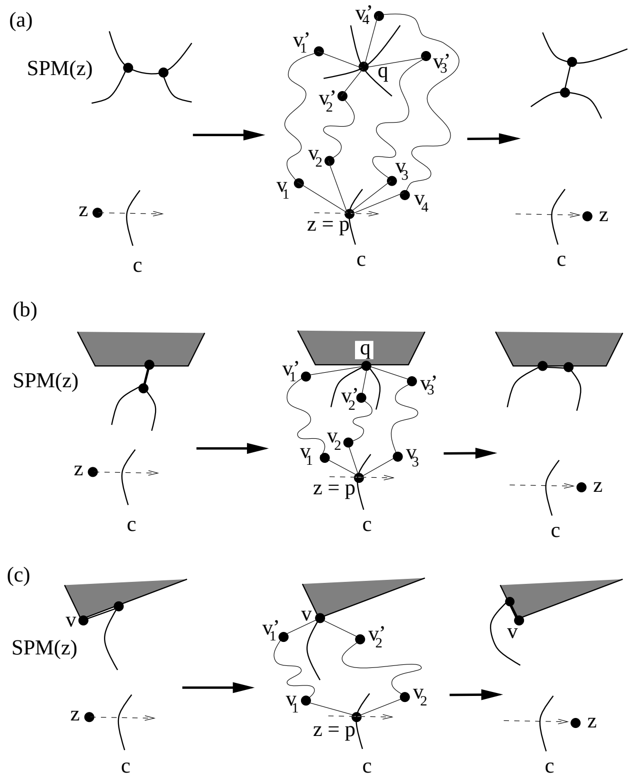

Figure 2: Three types of events that cause an edge contraction/expansion in $SPM(z)$ and thus define a topology curve c.

Consider first the type (1) topology curves (Fig. 2(a)). A type (1) topology curve c can be characterized as the locus of points $p \in P$ such that there exists a corresponding point q having four equidistant distinct shortest paths between p and q. If v_1, v_2, v_3 and v_4 are the vertices of P adjacent to p in the four shortest paths from p to q, then point p satisfies $|\overline{pv_1}| + d(v_1, q) = |\overline{pv_2}| + d(v_2, q) = |\overline{pv_3}| + d(v_3, q) = |\overline{pv_4}| + d(v_4, q)$.

If we substitute the coordinates, $p = (x_p, y_p)$ and $q = (x_q, y_q)$, into the above equations, we obtain three independent constraints in the four variables x_p, y_p, x_q and y_q, resulting in one degree of freedom; this yields an algebraic description of curve c. However, to complete the explicit set of equations, we must express $d(v_i, q)$ directly in terms of the variables, for all $i = 1, \ldots, 4$. For this, we assume that the vertex of P adjacent to q in the shortest path $\pi(q, v_i)$ is the vertex v_i', so that $d(v_i, q) = d(v_i, v_i') + |\overline{v_i' q}|$. (With a slight abuse of our terminology, we call v_i (resp. v_i') the *predecessor vertex* of p (resp. of q) in the shortest path from p to q through v_i and v_i'.) Then, a complete description of c is given by the equations

$$\begin{aligned} |\overline{pv_1}| + d(v_1, v_1') + |\overline{v_1' q}| &= |\overline{pv_2}| + d(v_2, v_2') + |\overline{v_2' q}| \\ &= |\overline{pv_3}| + d(v_3, v_3') + |\overline{v_3' q}| \\ &= |\overline{pv_4}| + d(v_4, v_4') + |\overline{v_4' q}|, \end{aligned}$$

where $d(v_i, v_i')$ is a constant, easily tabulated when we compute the shortest path maps $SPM(v_i)$, for $i = 1, \ldots, 4$.

We derive an upper bound on the number of the type (1) topology curves, c. A naive bound of $O(n^8)$ results from just considering the number of possible choices for the eight vertices, v_i and v_i', $i = 1, \ldots, 4$, used to specify c. We improve this bound as follows. After we choose the four vertices $v_1, \ldots, v_4$, we overlay the four shortest path maps $SPM(v_1), \ldots SPM(v_4)$, resulting in an arrangement of $O(n^2)$ cells. For a point q in any of the $O(n^2)$ cells, the shortest path from q to v_i has a fixed predecessor v_i', $i = 1, \ldots, 4$; this v_i' is the predecessor vertex of the cell of $SPM(v_i)$ in which q lies. Thus for a fixed set of predecessor vertices $v_1, \ldots, v_4$ of p, there are only $O(n^2)$ sets of vertices $v_1', \ldots, v_4'$ rather than $\binom{n-4}{4} = O(n^4)$. Therefore the total number of the type (1) topology curves is bounded by $\binom{n}{4} \cdot O(n^2) = O(n^6)$. An upper bound for the number of the type (2) (or (2)') topology curves can be derived similarly (see the full paper). We conclude that there are $O(n^6)$ topology curves in total.

Now, the complexity of the equivalence decomposition $\mathcal{A}$ is linear in the number of intersection points between pairs of curves that define it. There are three types of intersections: bisector-bisector, bisector-topology, and topology-topology curve intersections. Since any SPM has $O(n)$ bisectors, the n SPM's $(SPM(v))$ give $O(n^2)$ bisector curves, and thus there are $O(n^4)$ bisector-bisector intersections. The $O(n^6)$ topology curves and the $O(n^2)$ bisector curves give $O(n^8)$ bisector-topology intersections. To count the number of topology-topology intersections, we observe that there are three kinds of intersections between two topology curves: both curves are of type (1); both are of type (2); one is of type (1) and the other is of type (2).

We first discuss the case in which the intersections are between two type (1) topology curves. Consider the equations that define a type (1) curve c: for a point $p \in c$ there is a corresponding point q such that there are four distinct shortest paths of the same length from p to q, respectively via some predecessor vertices v_1 (and v_1'), v_2 (and v_2'), v_3 (and v_3'), and v_4 (and v_4'), where v_i and v_i' are respectively the predecessors of p and of q for $i = 1, \cdots, 4$. Now, consider another type (1) curve c' that intersects c at point p: c' is determined similarly, namely the locus of points p such that four distinct shortest paths are of equal length from p to *some point* q' (possibly *different from* q), via some predecessor vertices u_1 (and u'_1), u_2 (and u'_2), u_3 (and u'_3), and u_4 (and u'_4). There are six variables: $x_p, y_p, x_q, y_q, x_{q'}$, and $y_{q'}$, where $p = (x_p, y_p)$ and similarly for q and q'. We also have six equations: $d(p,q,v_1,v'_1) = d(p,q,v_2,v'_2) = d(p,q,v_3,v'_3) = d(p,q,v_4,v'_4)$, together with $d(p,q',u_1,u'_1) = d(p,q',u_2,u'_2) = d(p,q',u_3,u'_3) = d(p,q',u_4,u'_4)$, where we let $d(p,q,v_1,v'_1)$ denote the distance of the shortest path from p to q via v_1 and v'_1, etc. These six equations completely determine the six variables, and thus determine the intersection(s) between the curves c and c'. As in our derivation of the $O(n^6)$ bound on the number of topology curves, we have $\binom{n}{8} = O(n^8)$ ways to choose the eight vertices $v_1, \cdots, v_4, u_1, \cdots, u_4$. We then overlay the eight shortest path maps $SPM(v_i)$ and $SPM(u_i)$, $i = 1, \cdots, 4$, to result in an arrangement of $O(n^2)$ cells; this gives $O(n^2)$ choices for the predecessor vertices v'_i and u'_i, $i = 1, \cdots, 4$. Therefore, the total number of the type (1)-type (1) intersections is $O(n^8 \cdot n^2) = O(n^{10})$. (This is to be compared with the naive bound of $\binom{n}{16} = O(n^{16})$ or $O(n^6 \cdot n^6) = O(n^{12})$.) By a similar argument, we can show that the total number of the type (2)-type (2) intersections is $\binom{n}{6} \cdot O(n^2) \cdot O(n^2) = O(n^{10})$ (there are $O(n^2)$ choices of the two obstacle edges where q and q' lie), and that the total number of the type (1)-type (2) intersections is $\binom{n}{7} \cdot O(n^2) \cdot O(n) = O(n^{10})$. Therefore, we have $O(n^{10})$ topology-topology intersections in total.

LEMMA 4.1. *The complexity of the equivalence decomposition $\mathcal{A}$ is $O(n^{10})$.*

COROLLARY 4.1. *There are at most $O(n^{10})$ combinatorially distinct shortest path maps $SPM(z)$ for z in a polygonal domain having n vertices.*

4.2 Parametric Point Location. In this section we present our parametric point location data structure $\mathcal{D}(\sigma)$ for $SPM(\sigma)$, for each cell $\sigma \in \mathcal{A}$. The main challenge is that the structure $\mathcal{D}(\sigma)$ should depend only on the *topology* and not on the *geometry*, of $SPM(\sigma)$. Unfortunately, none of the existing point location data structures in the literature fulfills this requirement. Our solution is a new *optimal* point location method on SPM's that makes use of the properties of SPM's and is very simple. We also make the method *dynamic* for *topological updates* (namely contractions and expansions of edges in a SPM), and store the updates into a *persistent data structure* to save the overall space and preprocessing time, at the cost of slightly increasing the query time.

The idea of parametric point location is to store with each partitioning object in $\mathcal{D}(\sigma)$ its equation so that its exact position can be computed in $O(1)$ time given a source point $z \in \sigma$. Then the query algorithm proceeds as usual, except that during the search, each time we need a comparison between the query point q and an partitioning object, we compute the position of that object first.

Existing point location data structures rely on the geometry of $SPM(\sigma)$ and cannot serve for our purpose. The full paper discusses the deficiencies of several prior data structures.

We have devised a new point location method for $SPM(\sigma)$, which exploits special structure of the shortest path map. Roughly speaking, we do the following. First, we perform point location within a (fixed) subdivision induced by a shortest path tree rooted at a fixed point $z_0 \in \sigma$; this gives partial information on the location within $SPM(\sigma)$. Then, we complete the task of locating the query point within the set of bisector arcs, utilizing a centroid decomposition tree associated with the tree of bisector curves that lies within each face of the decomposition induced by $SPT(z_0)$. Details of the method, as well as the underlying structural results, are given in the full paper.

THEOREM 4.1. *Given a polygonal domain having n vertices, there exists a data structure using $O(n^{11})$ space and preprocessing time that supports two-point shortest path queries in $O(\log n)$ time.*

We also devise a method, employing persistent data structures, which allows us to reduce the space and preprocessing time, at the cost of slightly increasing the query time. In the full paper, we prove:

THEOREM 4.2. *Given a polygonal domain having n vertices, there exists a data structure using $O(n^{10} \log n)$ space and preprocessing time that supports two-point shortest path queries in $O(\log^2 n)$ time.*

5 Exploiting Visibility and Corridor Structure

In this section, we develop methods that utilize substantially smaller data structures, especially when the number h of holes in P is small compared with n. The tradeoff for this improvement in space complexity is that these methods have worst-case query time $\Omega(n)$ or $\Omega(n \log n)$ when the number of holes is very high (e.g., if $h = \Omega(n)$).

Visibility Structure. In the full paper, we describe a visibility-based method that results in the theorem below, which utilizes the notion of a "pivotal" vertex: We say that a vertex v is *pivotal* with respect to a given cell σ of $\mathcal{A}^{SPT}$, if for any point $z \in \sigma$, v is the first vertex on the (shortest) path, within $SPT(z)$, from z to some obstacle vertex v' *not* on the same obstacle as v.

THEOREM 5.1. *Using $O(n^5)$ time and space, one can compute a data structure that supports $O(\log n + \min\{h_s, h_t\})$ query time, where h_s (resp., h_t) is the number of pivotal obstacle vertices visible from s (resp., t). Here, h_s and h_t are bounded by $O(h)$.*

Exploiting Corridor Structure. We describe now a method to obtain *linear* in n space complexity, with logarithmic query time, for any fixed h. The method relies on the decomposition of the multiply-connected domain P into "corridors" ([15, 21]), which we now review.

First, we triangulate P; by standard techniques, this can be done in time $O(n \log n)$, but it can also be performed in nearly-optimal time $O(n + h \log^{1+\epsilon} h)$ [3]. Let $\mathcal{T}$ denote the resulting triangulation; we *fix* $\mathcal{T}$ in the following discussion. Let $\mathcal{G}_{\mathcal{T}}$ denote the graph-theoretic dual of $\mathcal{T}$. Then $\mathcal{G}_{\mathcal{T}}$ is a planar graph having $O(n)$ nodes, $O(n)$ arcs, and $h+1$ faces.

We now perform the following operations on $\mathcal{G}_{\mathcal{T}}$. First, we delete any degree-1 node of $\mathcal{G}_{\mathcal{T}}$, along with its incident edge; we continue doing this until there are no degree-1 nodes. At this stage, $\mathcal{G}_{\mathcal{T}}$ has $h+1$ faces and all nodes are of degree 2 or 3. We assume from now on that $h \geq 2$; the case $h \leq 1$ is easily handled separately (using an extension to [11] that we prove in the full paper). Thus, not all nodes are of degree 2, implying that there are at least two degree-3 nodes (since there must always be an even number of odd-degree vertices in a graph). Next, for each degree-2 node, we delete it and replace its 2 incident edges with a single edge. The resulting graph, call it $\mathcal{G}$, is a 3-regular planar graph, possibly with loops and possibly with multi-edges (2 edges joining the same pair of nodes). Further, $\mathcal{G}$ has $h+1$ faces, $2h-2$ nodes, and $3h-3$ arcs (by Euler's formula). The nodes of $\mathcal{G}$ correspond to triangles in $\mathcal{T}$, called *junction triangles.* If we remove the junction triangles from P, we are left with a set of polygons (one per arc of $\mathcal{G}$), called the *corridors* of P. The contraction process implies that the corridors are in fact *simple* polygons.

Now, the boundary of a corridor C consists of four portions: (1) a polygonal chain along ∂O_1, from a vertex a to a vertex b; (2) a diagonal $\overline{bc}$ from b to a vertex $c \in \partial O_2$ (possibly $O_2 = O_1$); (3) a polygonal chain along ∂O_2, from c to a vertex d; and (4) a diagonal $\overline{da}$. The segments $\overline{ad}$ and $\overline{bc}$ are called the *doors* of C; they separate C from adjacent junction triangles. (It may be that $a = b$ or that $c = d$, if C corresponds to a loop arc in $\mathcal{G}_{\mathcal{T}}$.) The (connected) region $H \subseteq C$ bounded by $\pi(a,b)$, $\overline{bc}$, $\pi(c,d)$, and $\overline{da}$ is called the *hourglass* [11] associated with C. The set difference $C \setminus H$ consists of a union of simple polygons called the *pockets* of C; each pocket has a *lid* consisting of a subpath (possibly a single segment) of one of the two shortest paths that determine H. We let Q denote the union of the junction triangles and hourglasses of P.

Each H may be an *open hourglass*, if $\pi(a,b) \cap \pi(c,d) = \emptyset$, or a *closed hourglass*, if $\pi(a,b) \cap \pi(c,d) = \pi(u,v)$, the *corridor path* linking the apex u of a *funnel* (with *base* $\overline{ad}$) to the apex v of a *funnel* (with *base* $\overline{bc}$). An open hourglass has two associated *convex chains*, $\pi(a,b)$ and $\pi(c,d)$. A closed hourglass has four associated convex chains: $\pi(a,u)$, $\pi(u,d)$, $\pi(b,v)$, and $\pi(v,c)$. In total time $O(n)$, after triangulation, we can identify the corridors and compute their associated hourglasses, pockets, and convex chains; this gives us also a full description of the region Q.

We now introduce the notion of a *coarsened shortest path map*, $CSPM(z)$. In its most general form, $CSPM(z)$ is defined with respect to a partitioning of a subset $V' \subseteq V$ of the vertex set V: $V' = V_1 \cup V_2 \cup \cdots \cup V_m$. The *cell*, $\sigma(V_i)$, corresponding to a set V_i of vertices is the locus of all points $p \in P$ for which the last vertex along a shortest path $\pi(z,p)$ is a vertex $r \in V_i$; we say that V_i is the *root* of cell $\sigma(V_i)$. Cells can share boundary points but must have pairwise-disjoint interiors. The cells $\sigma(V_i)$, in general, will not cover P; we let $\sigma(z) = P \setminus \cup_i \sigma(V_i)$. In the usual definition of a shortest path map, $V' = V$ and the sets V_i are the singleton vertices. Also, in general, the cells $\sigma(V_i)$ may be disconnected, consisting of many multiply-connected components. However, for our method here we use a particular choice of V' and its partitioning, which will enable us to conclude that the cells are simply connected and that they cover all of Q.

Consider a convex chain ξ. For a vertex $v \in \xi$, we say that a line segment $\overline{pv}$, for $p \in P$, is a *left tangency* (resp., *right tangency*) if the ξ lies in the (closed) halfplane to the left (resp., right) of the oriented line through $\overline{pv}$. For a given source point z on some convex chain, we say that $v \in \xi$ is a *left tangency vertex* (resp., *right tangency vertex*) if there exists a point $p \in P$ such that $\overline{pv}$ is a left tangency (resp., right tangency) and $\overline{pv} \subseteq \pi(z,p)$. It is not hard to see that

LEMMA 5.1. *For a fixed source point z on some convex chain, each vertex $v \in \xi$ on convex chain ξ is either a right tangency vertex or a left tangency vertex, not both.*

Furthermore, each convex chain ξ can be split in two according to the local direction of shortest paths whose last vertex lies on the chain:

LEMMA 5.2. *Consider a source point z on a convex chain of P. Let $\xi = (v_1, \ldots, v_m)$ be a convex chain of P, oriented so that ξ is rightward turning (v_{i+1} lies to the right of the oriented line through $\overline{v_{i-1}v_i}$). Then,* *there exists an index $j \in [0, m+1]$ such that $\{v_1, \ldots, v_j\}$ are left tangency vertices and $\{v_{j+1}, \ldots, v_m\}$ are right tangency vertices. (If $j = 0$ (resp., $j = m+1$), then all vertices are right (resp., left) tangency vertices.)*

Thus, by the above lemma, for a fixed z we can split each chain in at most two subchains according to identity of the vertices as left or right tangencies. How to perform this split is easily determined from the shortest path map $SPM(z)$ in linear time. Let $\mathcal{C}(z)$ denote the resulting set of convex chains.

We now define a particular coarsened shortest path map, $CSPM(z)$, as follows. Let V' be the set of obstacle vertices that lie on the convex chains $\mathcal{C}(z)$. Partition V' according to the $O(h)$ elements of $\mathcal{C}(z)$. In the full paper we prove:

LEMMA 5.3. *In the coarsened shortest path map $CSMP(z)$ based on the convex chains $\xi \in \mathcal{C}(z)$, the cells $\sigma(\xi)$ are simply connected and cover Q.*

In order to construct $CSPM(z)$, we first build $SPM(z)$. We then make a pass over the $SPM(z)$ and remove any bisector (extension segment) that bisects between two root vertices that are on the same convex chain or between two non-chain vertices (i.e., pocket vertices). The cells in the resulting subdivision of P correspond either to points having a root set V_i consisting of vertices along a single convex chain, or to points lying within a subpocket, having a vertex on the boundary of an adjacent cell that is rooted at a convex chain.

Since there are only $O(h)$ simply-connected cells, we know that the $CSPM(z)$ has only $O(h)$ nodes of degree greater than two in its set of edges. However, the boundary between two cells of $CSPM(z)$ may have complexity $\Omega(n)$, if the root chains have complexity $\Omega(n)$. It is important for our space bound that we *simplify* the $CSPM(z)$, into a new type of subdivision, which we call a *simplified coarsened shortest path map*, so that it can be stored using only $O(h)$ line segments (in addition to the $O(n)$ boundary complexity of P, which is constant over all choices of z). This is done as follows. First, we note that the bisector in $CSPM(z)$ between two chains ξ and ξ' is, in general, a curve consisting of many hyperbolic arcs joined end-to-end. We *delete* all such bisecting curves (as they may have complexity $\Omega(n)$), but add a line segment joining each of the $O(h)$ degree-3 nodes (that occur where three such bisecting curves come together) to each of their three root vertices along the three (distinct) convex chains that serve as the common "root" of the node. We are left with $O(h)$ segments, S, that partition P into $O(h)$ regions. By deleting bisecting chains, we have created ambiguity: By locating a point in the new simplified CSPM, we no longer know the exact identity of the root chain of that point — there are *two* possible root chains. But from the point of view of determining an optimal path to a query point, this does not matter: We can compute both paths and compare their lengths. When locating a point in the resulting simplified CSPM, though, we do have one more step to perform in order to compute the shortest path length: We must find the point of tangency from the query point to the root convex chain. This is easily done in $O(\log n)$ time, by binary search. (Further details are given in the full paper.)

THEOREM 5.2. *One can compute a data structure of size $O(n + h^5)$ supporting $O(h \log n)$-time two-point shortest path queries in a polygonal domain having n vertices and h holes.*

6 Queries on Polyhedral Surfaces

One extension of our results is to the problem of two-point shortest path queries on nonconvex surfaces. By a generalization of our methods, we can construct data structures that are a factor of n larger than those we construct in a planar polygon, P, while achieving the same polylogarithmic query times. The main idea is to consider each of the n facets separately, and to construct decompositions of each, using "virtual" vertex sources that have been unfolded into the plane of the facet (see [1]). Details will appear in the full paper.

7 Conclusion

We have offered some of the first algorithmic solutions to the exact two-point shortest path query problem. It is probably possible to improve some of our space and time bounds. It would be most interesting to see if one can achieve, for instance, sublinear query time, while using only quadratic space (e.g., storing the set of shortest path maps rooted at every vertex).

An interesting combinatorial question is also suggested by our work: *How many combinatorially distinct shortest path maps are there for a polygonal domain having n vertices?* Our results have shown that this number is somewhere between $\Omega(n^4)$ and $O(n^{10})$, leaving a rather large gap. A related open question is to determine the combinatorial complexity of the lower envelope $g = \min_i g_i(x_1, y_1, x_2, y_2)$, for the functions g_i defined in Section 3; we know of nothing better than the $O(n^{12+\epsilon})$ upper bound implied by the fact that the lower envelope has complexity $O(n^{4+\epsilon})$ for each of the $O(n^8)$ choices of pairs of cells in $\mathcal{A}^{SPT}$.

Also, although our emphasis here has been on *exact* query methods, it would be most interesting to see if one can perform $O(1)$-approximate two-point queries in polylogarithmic time, using nearly linear storage.

Acknowledgements. We thank Sariel Har-Peled and Micha Sharir for their input on the subject of this paper.

References

[1] P. K. Agarwal, B. Aronov, J. O'Rourke, and C. A. Schevon. Star unfolding of a polytope with applications. *SIAM J. Comput.*, 26:1689–1713, 1997.

[2] S. R. Arikati, D. Z. Chen, L. P. Chew, G. Das, M. H. M. Smid, and C. D. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. *4th Annual European Symposium*, Springer LNCS Vol. 1136, 1996.

[3] R. Bar-Yehuda and B. Chazelle. Triangulating disjoint Jordan chains. *Internat. J. Comput. Geom. Appl.*, 4(4):475–481, 1994.

[4] B. Chazelle. A theorem on polygon cutting with applications. In *Proc. 23rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 339–349, 1982.

[5] B. Chazelle and L. J. Guibas. Visibility and intersection problems in plane geometry. *Discrete Comput. Geom.*, 4:551–581, 1989.

[6] D. Z. Chen. On the all-pairs Euclidean short path problem. In *Proc. 6th ACM-SIAM Sympos. Discrete Algorithms*, pages 292–301, 1995.

[7] D. Z. Chen, O. Daescu, and K. S. Klenk. On geometric path query problems. In *Proc. 5th Workshop Algorithms Data Struct.*, volume 1272 of *Lecture Notes Comput. Sci.*, pages 248–257. Springer-Verlag, 1997.

[8] K. L. Clarkson. Approximation algorithms for shortest path motion planning. In *Proc. 19th Annu. ACM Sympos. Theory Comput.*, pages 56–65, 1987.

[9] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.*, 15(2):317–340, 1986.

[10] M. Goodrich and R. Tamassia. Dynamic trees and dynamic point location. In *Proc. 23rd Annu. ACM Sympos. Theory Comput.*, pages 523–533, 1991.

[11] L. J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. *J. Comput. Syst. Sci.*, 39:126–152, 1989.

[12] S. Har-Peled. Approximate shortest paths and geodesic diameters on convex polytopes in three dimensions. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 359–365, 1997.

[13] P. J. Heffernan and J. S. B. Mitchell. An optimal algorithm for computing visibility in the plane. *SIAM J. Comput.*, 24(1):184–201, 1995.

[14] J. Hershberger and S. Suri. An optimal algorithm for Euclidean shortest paths in the plane. Manuscript, Washington University, 1995.

[15] S. Kapoor, S. N. Maheshwari, and J. S. B. Mitchell. An efficient algorithm for Euclidean shortest paths among polygonal obstacles in the plane. *Discrete Comput. Geom.*, 18:377–383, 1997.

[16] D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12(1):28–35, 1983.

[17] J. S. B. Mitchell. Shortest paths among obstacles in the plane. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 308–317, 1993.

[18] J. S. B. Mitchell. Shortest paths among obstacles in the plane. *Internat. J. Comput. Geom. Appl.*, 6:309–332, 1996.

[19] J. S. B. Mitchell. Shortest paths and networks. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 24, pages 445–466. CRC Press LLC, Boca Raton, FL, 1997.

[20] J. S. B. Mitchell. Geometric shortest paths and network optimization. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, page ?? Elsevier Science Publishers B.V. North-Holland, Amsterdam, 1998.

[21] J. S. B. Mitchell and S. Suri. Separation and approximation of polyhedral objects. *Comput. Geom. Theory Appl.*, 5:95–114, 1995.

[22] M. Pocchiola and G. Vegter. Computing the visibility graph via pseudo-triangulations. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 248–257, 1995.

[23] F. P. Preparata and R. Tamassia. Efficient point location in a convex spatial cell-complex. *SIAM J. Comput.*, 21:267–280, 1992.

[24] N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Commun. ACM*, 29:669–679, 1986.

[25] M. Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications.* Cambridge University Press, New York, 1995.

[26] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–381, 1983.

[27] S. Suri. Polygons. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 23, pages 429–444. CRC Press LLC, Boca Raton, FL, 1997.