L_1 Shortest Path Queries among Polygonal Obstacles in the Plane

Danny Z. Chen[*1] **and Haitao Wang**[†2]

1 **Department of Computer Science and Engineering**
University of Notre Dame, Notre Dame, IN 46556, USA
dchen@cse.nd.edu

2 **Department of Computer Science, Utah State University**
Logan, UT 84322, USA
haitao.wang@usu.edu

Abstract

Given a point s and a set of h pairwise disjoint polygonal obstacles with a total of n vertices in the plane, after the free space is triangulated, we present an $O(n + h \log h)$ time and $O(n)$ space algorithm for building a data structure (called *shortest path map*) of size $O(n)$ such that for any query point t, the length of the L_1 shortest obstacle-avoiding path from s to t can be reported in $O(\log n)$ time and the actual path can be found in additional time proportional to the number of edges of the path. Previously, the best algorithm computes such a shortest path map in $O(n \log n)$ time and $O(n)$ space. In addition, our techniques also yield an improved algorithm for computing the L_1 geodesic Voronoi diagram of m point sites among the obstacles.

1998 ACM Subject Classification F.2 Analysis of algorithms and problem complexity

Keywords and phrases computational geometry, shortest path queries, shortest paths among obstacles, L_1/L_∞/rectilinear metric, shortest path maps, geodesic Voronoi diagrams

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.293

1 Introduction

Given a point s and a set of h pairwise disjoint polygonal obstacles, $\mathcal{P} = \{P_1, P_2, \ldots, P_h\}$, with a total of n vertices in the plane, where s is considered as a special point obstacle, the plane minus the interior of the obstacles is called the *free space* of $\mathcal{P}$. Two obstacles are pairwise *disjoint* if they do not intersect in their interior. The L_1 *shortest path query problem*, denoted by L_1-SPQ, is to compute a data structure (called *shortest path map* or SPM for short) with s as the *source point* such that for any query point t, an L_1 shortest obstacle-avoiding path from s to t can be obtained efficiently. Note that such a path can have any polygonal segments but the length of each segment of the path is measured by the L_1 metric. Unless otherwise stated, all SPMs mentioned in this paper have the following performances: for any query point t, the length of the L_1 shortest path from s to t can be reported in $O(\log n)$ time and the actual path can be found in additional time proportional to the number of edges of the path.

We also study the L_1 *geodesic Voronoi diagram problem*, denoted by L_1-GVD: Given an obstacle set $\mathcal{P}$ and a set of m point sites in the free space, compute the geodesic Voronoi diagram for the m point sites under the L_1 distance metric among the obstacles in $\mathcal{P}$.

* The research of Chen was supported in part by NSF under Grants CCF-0916606 and CCF-1217906.
† Corresponding author.

30th Symposium on Theoretical Aspects of Computer Science (STACS'13).
Editors: Natacha Portier and Thomas Wilke; pp. 293–304
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

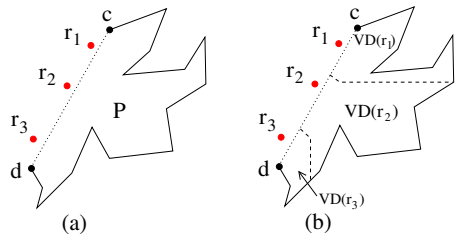

Figure 1 (a) Three weighted point sites (r_1, r_2, r_3) and a simple polygon P with an open edge $\overline{cd}$. The goal is to compute the L_1 geodesic Voronoi diagram in P for the three sites which influence P only through the edge $\overline{cd}$. (b) Illustrating a possible solution: P is partitioned into three Voronoi regions $VD(r_i)$ for each r_i, $1 \leq i \leq 3$.

Computing L_1 shortest paths has been studied extensively (e.g., see [5, 8, 15, 18, 19, 21]). Mitchell [18, 19] builds an SPM in $O(n \log n)$ time and $O(n)$ space, which is optimal when $h = \Theta(n)$ because finding an L_1 shortest path has a lower bound of $\Omega(n + h \log h)$ on the running time [9]. Throughout this paper, let T refer to the time for triangulating the free space of $\mathcal{P}$ and let $\epsilon > 0$ be any arbitrarily small constant. It is known that $T = O(n \log n)$ and $T = O(n + h \log^{1+\epsilon} h)$ [1]. Recently, Chen and Wang gave an algorithm that can find a single shortest path in $O(T + n + h \log h)$ time and $O(n)$ space [5]. However, the algorithm in [5] cannot construct an SPM, which is left as an open problem in [5]. For L_1-GVD, Mitchell's algorithm [18, 19] can be extended to solve it in $O((n + m) \log(n + m))$ time.

1.1 Our Results

In this paper, we answer the open problem in [5] by presenting an algorithm that builds an SPM of size $O(n)$ in $O(n + h \log h)$ time and $O(n)$ space after the free space is triangulated in $O(T)$ time. Hence, the running time of the overall algorithm is $O(T + n + h \log h)$. If the triangulation can be done optimally (i.e., $T = O(n + h \log h)$), then our algorithm matches the $\Omega(n + h \log h)$ time lower bound [9]. In addition, it is easy to see that given an SPM, we can add $h - 1$ segments in the free space to connect the obstacles in $\mathcal{P}$ together to obtain a single simple polygon and then triangulate the free space, in totally $O(n)$ time [1, 2]. This shows that the problem L_1-SPQ is solvable in $\Theta(T)$ time, i.e., building an SPM is equivalent to triangulating the free space of $\mathcal{P}$ in terms of the running time.

As Mitchell's algorithm [18, 19], our techniques can also be extended to solve the L_1-GVD problem in $O(T' + (m + h) \log(m + h)))$ time, where T' is the time for triangulating the free space of $\mathcal{P}$ with the m point sites and $T' = \min\{O((n + m) \log(n + m)), O(n + (m + h) \log^{1+\epsilon}(m + h))\}$ [1]. Our new algorithm is faster than Mitchell's $O((n + m) \log(n + m))$ time solution for sufficiently small m and h (e.g., when $m + h = O(n^{1-\epsilon})$).

It is well known that with solutions in L_1 version, the same problems in the rectilinear version and L_∞ version can be solved immediately [18, 19]. Hence, our results also hold for the rectilinear version and the L_∞ version of the problems.

A *challenging subproblem* we need to solve is a special case of the (additively) weighted L_1 geodesic Voronoi diagram problem on a simple polygon P: The weighted point sites all lie outside P and influence P through an (open) edge (e.g., see Fig. 1). Our main effort of this paper is to solve this problem in $O(n' + m')$ time, where n' is the number of vertices of P and m' is the number of sites. This problem is interesting in its own right.

This subproblem may not look "challenging" at all as it can be solved by many existing techniques, e.g., the continuous Dijkstra paradigm [18, 19], the sweeping algorithm [11], and divide-and-conquer [20]. However, all these methods would lead to an $O((n'+m') \log(n'+m'))$

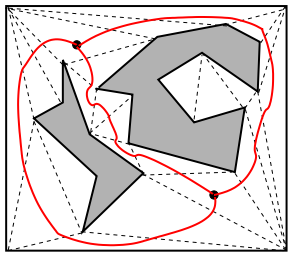

Figure 2 Illustrating a triangulation of the free space among two obstacles and the corridors (with red solid curves). There are two junction triangles indicated by the large dots inside them, connected by three solid (red) curves. Removing the two junction triangles results in three corridors.

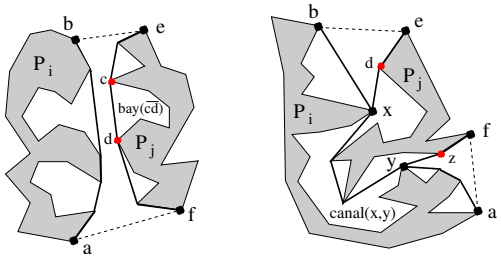

Figure 3 Illustrating an open hourglass (left) and a closed hourglass (right) with a corridor path connecting the apices x and y of the two funnels. The dashed segments are diagonals. The paths $\pi(a, b)$ and $\pi(e, f)$ are shown with thick solid curves. A bay $bay(\overline{cd})$ with gate $\overline{cd}$ (left) and a canal $canal(x, y)$ with gates $\overline{xd}$ and $\overline{yz}$ (right) are also indicated.

time solution, and consequently, the overall time for building an SPM would be $O(n \log n)$. Our linear time algorithm can be viewed as an incremental approach. Incremental approaches have been widely used in geometric algorithms, and normally they can result in good randomized algorithms. Incremental approaches have also been used for constructing Voronoi diagrams, which usually take quadratic time. Our result demonstrates that incremental approaches are able to yield optimal deterministic solutions for building Voronoi diagrams. The new techniques we provide here should be generalized to solving other related problems.

For simplicity of discussion, as in [18, 19], we assume no two obstacle vertices lie on the same horizontal or vertical line. Henceforth, unless otherwise stated, a shortest path refers to an L_1 shortest path and a length is in the L_1 metric.

In the following, in Section 2, we review some geometric structures. In Section 3, we outline our algorithm for computing an SPM. Particularly, our algorithm for solving the challenging subproblem is in Section 4. Due to the space limit, many details (including the algorithm for L_1-GVD) are omitted and can be found in the full version of this paper [6].

2 Preliminaries

In this section, we review some geometric structures of $\mathcal{P}$, i.e., the corridors, ocean, bays, and canals, which have been used previously, e.g., [5, 7, 16].

For simplicity, we assume all obstacles are contained in a rectangle $\mathcal{R}$ (see Fig. 2). We also view $\mathcal{R}$ as an obstacle in $\mathcal{P}$. Let $\mathcal{F}$ be the free space inside $\mathcal{R}$. We compute an arbitrary triangulation of $\mathcal{F}$, denoted by $Tri(\mathcal{F})$, in $O(T)$ time.

Let $G(\mathcal{F})$ be the (planar) dual graph of $Tri(\mathcal{F})$. As shown in [16], based on $G(\mathcal{F})$, we compute a 3-regular graph, denoted by G^3 (the degree of every node in G^3 is three), possibly with loops and multi-edges, as follows. First, remove every degree-one node from $G(\mathcal{F})$ along with its incident edge; repeat this process until no degree-one node remains. Second, remove every degree-two node from $G(\mathcal{F})$ and replace its two incident edges by a single edge; repeat this process until no degree-two node remains. The resulting graph is G^3 (see Fig. 2), which has $O(h)$ faces, $O(h)$ nodes, and $O(h)$ edges [16]. Each node of G^3 corresponds to a triangle in $Tri(\mathcal{F})$, which is called a *junction triangle* (see Fig. 2). The removal of all junction triangles from $Tri(\mathcal{F})$ results in $O(h)$ *corridors*, each of which corresponds to one edge of G^3.

The boundary of a corridor C consists of four parts (see Fig. 3): (1) A boundary portion of an obstacle $P_i \in \mathcal{P}$, from a point a to a point b; (2) a diagonal of a junction triangle from

b to a boundary point e on an obstacle $P_j \in \mathcal{P}$ ($P_i = P_j$ is possible); (3) a boundary portion of the obstacle P_j from e to a point f; (4) a diagonal of a junction triangle from f to a. Let $\pi(a, b)$ (resp., $\pi(e, f)$) be the shortest path from a to b (resp., e to f) inside C. The region H_C bounded by $\pi(a, b), \pi(e, f)$, and the two diagonals $\overline{be}$ and $\overline{fa}$ is called an *hourglass*, which is *open* if $\pi(a, b) \cap \pi(e, f) = \emptyset$ and *closed* otherwise (see Fig. 3). If H_C is open, then $\pi(a, b)$ and $\pi(e, f)$ are called *sides* of H_C; otherwise H_C consists of two "funnels" and a path $\pi_C = \pi(a, b) \cap \pi(e, f)$ joining the two apices of the two funnels, called the *corridor path* of C. The two funnel apices connected by the corridor path are called the *corridor path terminals*.

Let $\mathcal{M}$ be the union of all $O(h)$ junction triangles, open hourglasses, and funnels. We call $\mathcal{M}$ the *ocean*. Denote by $SPM(\mathcal{F})$ the SPM we want to compute on the free space $\mathcal{F}$ (with respect to the source point s), and denote by $SPM(\mathcal{M})$ the portion of $SPM(\mathcal{F})$ in $\mathcal{M}$. After the free space is triangulated in $O(T)$ time, the algorithm in [5] computes $SPM(\mathcal{M})$ of size $O(n)$ in $O(n + h \log h)$ time and $O(n)$ space, based on the following observation: For any point $t \in \mathcal{M}$, there exists a shortest s-t path $\pi(s, t)$ in $\mathcal{F}$ such that $\pi(s, t)$ is in $\mathcal{M}$ but possibly contains some corridor paths. Our task in this paper is to compute the portion of $SPM(\mathcal{F})$ in the space $\mathcal{F} \setminus \mathcal{M}$, in additional $O(n)$ time. Below, we partition the space $\mathcal{F} \setminus \mathcal{M}$ into two types of regions: *bays* and *canals*. Consider the hourglass H_C of a corridor C. Depending on whether H_C is open or closed, there are two cases.

If H_C is open (see Fig. 3), then H_C has two sides. Let $S_1(H_C)$ be an arbitrary side of H_C. The obstacle vertices on $S_1(H_C)$ all lie on the same obstacle, say $P \in \mathcal{P}$. Let c and d be any two adjacent vertices on $S_1(H_C)$ such that the line segment $\overline{cd}$ is not an edge of P (see the left figure in Fig. 3, with $P = P_j$). The region enclosed by $\overline{cd}$ and a boundary portion of P between c and d is called the *bay* of $\overline{cd}$ and P, denoted by $bay(\overline{cd})$, which is a simple polygon. We call $\overline{cd}$ the *bay gate* of $bay(\overline{cd})$, which is a common edge of $bay(\overline{cd})$ and $\mathcal{M}$.

If the hourglass H_C is closed, then let x and y be the two apices of its two funnels. Consider two adjacent vertices c and d on a side of a funnel such that the line segment $\overline{cd}$ is not an obstacle edge. If neither c nor d is a funnel apex, then c and d must both lie on the same obstacle and the segment $\overline{cd}$ also defines a bay with that obstacle. However, if either c or d is a funnel apex, say, $x = c$, then x and d may lie on different obstacles. If they lie on the same obstacle, then they also define a bay; otherwise, we call $\overline{xd}$ the *canal gate* at x (see Fig. 3). Similarly, there is also a canal gate at the other funnel apex y, say $\overline{yz}$. Let P_i and P_j be the two obstacles bounding the hourglass H_C. The obstacle-free region enclosed by P_i, P_j, and the two canal gates $\overline{xd}$ and $\overline{yz}$ that contain the corridor path of H_C is called the *canal* of H_C, denoted by $canal(x, y)$, which is a simple polygon. Similarly, the two canal gates are common edges of the canal and $\mathcal{M}$.

Clearly, all the bays and canals together constitute the space $\mathcal{F} \setminus \mathcal{M}$. Note that bays and canals are connected with $\mathcal{M}$ only through their gates.

3 The Algorithm Outline

Our task is to compute $SPM(\mathcal{F})$. To this end, again, $SPM(\mathcal{M})$ has already been computed in [5], our task in this paper is to compute the portion of $SPM(\mathcal{F})$ in $\mathcal{F} \setminus \mathcal{M}$, i.e., all bays and canals, or in other words, compute an SPM for each bay/canal. More intuitively, we "expand" $SPM(\mathcal{M})$ to all bays/canals through their gates, in additional $O(n)$ time.

Recall that an SPM is a partition of the free space into many cells; each cell C has a root point r such that for any point p in C, a shortest path from the source point s to p consists of the line segment $\overline{pr}$ and a shortest path from s to r. Further, $\overline{pr}$ is in C (i.e., C is a star-shaped polygon with r in the kernel). Refer to [18, 19] for more details on SPM.

We discuss the bays first. Consider a bay $bay(\overline{cd})$. Since its gate $\overline{cd}$ is also an edge of $\mathcal{M}$, $\overline{cd}$ is adjacent to some cells in $SPM(\mathcal{M})$. If $\overline{cd}$ is in a single cell $C(r)$ of $SPM(\mathcal{M})$ with r as the root, then each point in $bay(\overline{cd})$ has a shortest path to s via r. Thus, to construct an SPM for $bay(\overline{cd})$, it suffices to compute an SPM on $bay(\overline{cd})$ with respect to the point r, which can be done in linear time (in terms of the number of vertices of $bay(\overline{cd})$) since $bay(\overline{cd})$ is a simple polygon[1]. Note that although r may not be a point in $bay(\overline{cd})$, we can, for example, connect r to both c and d with two line segments to form a new simple polygon that contains $bay(\overline{cd})$.

If the gate $\overline{cd}$ is not contained in a single cell of $SPM(\mathcal{M})$, then $\overline{cd}$ intersects multiple cells in $SPM(\mathcal{M})$. When computing an SPM for $bay(\overline{cd})$, we must consider the roots of all such cells and each root has an additive weight that is the length of its shortest path to s. In this case, multiple vertices of $SPM(\mathcal{M})$ (i.e., the intersections of the boundaries of the cells of $SPM(\mathcal{M})$ with $\overline{cd}$) may lie in the interior of $\overline{cd}$. We call the vertices of $SPM(\mathcal{M})$ on $\overline{cd}$ (including its endpoints c and d) the $SPM(\mathcal{M})$ *vertices*. Later in Section 4, we give an algorithm for the following result.

▶ **Theorem 1.** *For a bay of n' vertices with m' $SPM(\mathcal{M})$ vertices on its gate, an SPM of size $O(n'+m')$ for the bay can be computed in $O(n'+m')$ time.*

Since a canal has two gates which are also edges of $\mathcal{M}$, multiple $SPM(\mathcal{M})$ vertices may lie on both its gates. Later in Section 5, we give an algorithm for the following Theorem 2, which uses the algorithm for Theorem 1 as a main procedure.

▶ **Theorem 2.** *For a canal of n' vertices with totally m' $SPM(\mathcal{M})$ vertices on its two gates, an SPM of size $O(n'+m')$ can be computed in $O(n'+m')$ time.*

By Theorems 1 and 2, the total time for computing the SPMs for all bays and canals is linear in terms of the total sum of the numbers of obstacle vertices of all bays and canals (which is $O(n)$) and the total number of the $SPM(\mathcal{M})$ vertices on the gates of all bays and canals (which is also $O(n)$ since the size of $SPM(\mathcal{M})$ is $O(n)$). We hence conclude that after $SPM(\mathcal{M})$ is obtained, an SPM for $\mathcal{F}$ can be computed in additional $O(n)$ time. Together with a planar point location data structure [10, 17], we have the following result.

▶ **Theorem 3.** *After the free space $\mathcal{F}$ is triangulated in $O(T)$ time, an $SPM(\mathcal{F})$ of size $O(n)$ can be built in $O(n+h\log h)$ time and $O(n)$ space.*

4 Expanding the $SPM(\mathcal{M})$ into a Bay (a Sketch)

In this section, we sketch our algorithm for Theorem 1, and all details can be found in [6].

Consider a bay $bay(\overline{cd})$ with gate $\overline{cd}$ (see Fig. 3). Denote by n' the number of vertices of $bay(\overline{cd})$. Let $SPM(bay(\overline{cd}))$ be an SPM for $bay(\overline{cd})$ that we seek to compute. If $\overline{cd}$ lies in a single cell of $SPM(\mathcal{M})$, we have shown $SPM(bay(\overline{cd}))$ can be computed in $O(n')$ time. This section focuses on the case when $\overline{cd}$ is not contained in a single cell of $SPM(\mathcal{M})$. Denote by m' the number of $SPM(\mathcal{M})$ vertices on $\overline{cd}$. Our task is to compute $SPM(bay(\overline{cd}))$ in $O(n'+m')$ time. Let R be the set of roots of the cells of $SPM(\mathcal{M})$ that intersect with $\overline{cd}$.

To obtain $SPM(bay(\overline{cd}))$, we first compute, for each $r \in R$, the *Voronoi region* $VD(r)$ inside $bay(\overline{cd})$ such that for any point $t \in VD(r)$, there is a shortest s-t path going through

[1] As the Euclidean shortest path between two points in a simple polygon is also an L_1 shortest path [13], a Euclidean SPM [12] in a simple polygon is also an L_1 one.

r; we then compute an SPM on $VD(r)$ with respect to the single point r, which can be done in linear time since $VD(r)$ is a simple polygon. Thus, the key is to decompose $bay(\overline{cd})$ into Voronoi regions for the roots of R (which is the challenging subproblem mentioned in Section 1.1). Denote by $VD(bay(\overline{cd}))$ this Voronoi diagram decomposition of $bay(\overline{cd})$. We aim to compute $VD(bay(\overline{cd}))$ in $O(n' + m')$ time.

Without loss of generality (W.l.o.g.), assume that $\overline{cd}$ is positive-sloped, $bay(\overline{cd})$ is on the right of $\overline{cd}$, and the vertex c is higher than d (e.g., $bay(\overline{cd}) = P$ in Fig. 1). Let $R = \{r_1, r_2, \ldots, r_k\}$ be the set of roots of the cells of $SPM(\mathcal{M})$ that intersect with $\overline{cd}$ in the order from c to d along $\overline{cd}$. Note that R may be a multi-set, i.e., two roots r_i and r_j with $i \neq j$ may refer to the same physical point; but this is not important to our algorithm (e.g., we can view each r_i as a physical copy of the same root). Let $c = v_0, v_1, \ldots, v_k = d$ be the $SPM(\mathcal{M})$ vertices on $\overline{cd}$ ordered from c to d (thus $m' = k + 1$). Hence, for each $1 \leq i \leq k$, the segment $\overline{v_{i-1}v_i}$ is on the boundary of the cell $C(r_i)$ of $SPM(\mathcal{M})$. To obtain $VD(bay(\overline{cd}))$, for each $r_i \in R$, we need to compute the Voronoi region $VD(r_i)$.

Our algorithm can be viewed as an incremental one, i.e., it considers the roots in R one by one. It is commonly known that incremental approaches can construct Voronoi diagrams in quadratic time, or may give good randomized results. In contrast, our algorithm is deterministic and takes only linear time. The success of it hinges on that we can find an *order* of the roots in R such that by following this order to consider the roots in R incrementally, we are able to compute $VD(bay(\overline{cd}))$ in linear time. The order is nothing but that of the indices of the roots in R we have defined. With this order, the algorithm is conceptually simple. However, it is quite challenging to argue its correctness and achieve a linear time implementation. Our strategy is to show that the algorithm implicitly maintains a number of *invariants* that assure the correctness of the algorithm. For this purpose, we discover many observations that capture some essential properties of this L_1 problem.

4.1 Algorithm Overview

To compute $VD(bay(\overline{cd}))$, it turns out that we need to deal with the interactions between some horizontal and vertical rays, each of which belongs to the bisector of two roots in R. Further, considering the roots in R incrementally is equivalent to considering the corresponding rays incrementally. We process these rays in a certain order (e.g., as to be proved, their origins somehow form a staircase structure). For each ray considered, if it is vertical, then it is easy (it eventually leads to a ray shooting operation), and its processing does not introduce any new ray. But, if it is horizontal, then the situation is more complicated since its processing may introduce many new horizontal rays and (at most) one vertical ray, also in a certain order along a staircase structure (in addition to causing a ray shooting operation). A stack is used to store certain vertical rays that need to be further processed.

The algorithm needs to perform ray shooting operations for some vertical and horizontal rays. Although there are known data structures for ray shooting queries [3, 4, 12, 14], they are not efficient enough for a linear time implementation of the entire algorithm. Based on observations, we use the horizontal visibility map and vertical visibility map of $bay(\overline{cd})$ [2]. More specifically, we prove that all vertical ray shootings are in a "nice" sorted order (called *target-sorted*). With this property, all vertical ray shootings are performed in totally linear time by using the vertical visibility map of $bay(\overline{cd})$. The horizontal visibility map is used to guide the overall process of the algorithm. During the algorithm, we march into the bay and the horizontal visibility map allows us to keep track of our current position (i.e., in a trapezoid of the map that contains our current position). The horizontal visibility map also allows each horizontal ray shooting to be done in $O(1)$ time. In addition, in the preprocessing

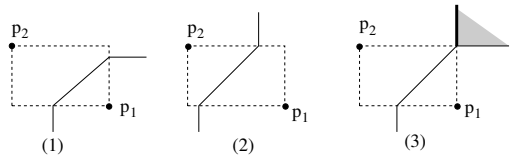

Figure 4 The L_1 bisector $B(p_1, p_2)$ of two weighted points p_1 and p_2. In (3), an entire quadrant (the shaded area) is $B(p_1, p_2)$, but we choose $B(p_1, p_2)$ to be the vertical (solid thick) half-line.

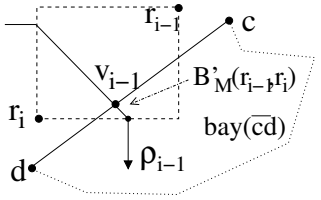

Figure 5 Illustrating the definition of ρ_{i-1}.

of the algorithm, we also need to perform some other ray shootings (for rays of slope -1); our linear time solution for this also hinges on the target-sorted property of such rays.

Our algorithm is conceptually simple. The only data structures we need are linked lists, a stack, and the horizontal and vertical visibility maps. Again, it is much more difficult to argue the correctness of the algorithm, making the presentation of this paper lengthy, technically complicated, or even tedious, for which we ask for the reader's patience.

Below, we sketch how our algorithm works, and the proof of its correctness is omitted. We may also use some terminology of natural meaning without definitions (their formal definitions are can also be found in [6].

4.2 The Algorithm

Each root $r_i \in R$ can be viewed as an additively weighted point whose weight is the length of an L_1 shortest path from s to r_i. For any two weighted points p_1 and p_2 with weights w_1 and w_2, respectively, their bisector $B(p_1, p_2)$ can be an entire quadrant of the plane (e.g., see Fig. 4); in this case, as in [18, 19], we choose a vertical half-line as the bisector. Denote by $Rec(p_1, p_2)$ the rectangle with p_1 and p_2 as its two diagonal vertices. Thus, as illustrated in Fig. 4, $B(p_1, p_2)$ consists of three portions, two rays whose origins are on the boundary of $Rec(p_1, p_2)$ and an open line segment (called *middle segment* and denoted by $B_M(p_1, p_2)$) in $Rec(p_1, p_2)$ connecting the origins the two rays; further each ray is either horizontal or vertical and the middle segment is of slope 1 or -1.

For any pair of consecutive roots r_{i-1} and r_i in R for $2 \leq i \leq k$, since v_{i-1} is on the common boundary of the cells $C(r_{i-1})$ and $C(r_i)$, v_{i-1} lies on the bisector $B(r_{i-1}, r_i)$ of r_{i-1} and r_i. But v_{i-1} may lie on either a ray or the middle segment of $B(r_{i-1}, r_i)$. If v_{i-1} lies on a ray of $B(r_{i-1}, r_i)$, let ρ_{i-1} denote the ray; otherwise, v_{i-1} partitions $B_M(r_{i-1}, r_i)$ into two portions and one portion (denoted by $B'_M(r_{i-1}, r_i)$) intersects the interior of $bay(\overline{cd})$, and we let ρ_{i-1} be the ray of $B(r_{i-1}, r_i)$ connecting to $B'_M(r_{i-1}, r_i)$ (see Fig. 5). In either case, ρ_{i-1} is either vertically going south (downwards) or horizontally going east (rightwards). (If $B_M(r_{i-1}, r_i)$ does not intersect $\overline{cd}$, we let $B'_M(r_{i-1}, r_i) = \emptyset$.) Further, if $v_{i-1} \in B_M(r_{i-1}, r_i)$, $B_M(r_{i-1}, r_i)$ must be (-1)-sloped [6]. Denote by $or(\rho)$ the origin of a ray ρ. We can prove that the origins $or(\rho_1), or(\rho_2), \ldots, or(\rho_{k-1})$ follow the order from *northeast* to *southwest* [6].

Let ∂ denote the boundary of $bay(\overline{cd})$ excluding $\overline{cd}$. The algorithm first determines for each $2 \leq i \leq k$ whether $B'_M(r_{i-1}, r_i)$ intersects ∂, which is done by a set of (-1)-sloped ray shootings. If, say $B'_M(r_{i-1}, r_i)$, intersects ∂ at a point p such that $\overline{v_{i-1}p}$ is inside $bay(\overline{cd})$, then $\overline{v_{i-1}p}$ partitions $bay(\overline{cd})$ into two simple polygons bay_1 and bay_2 such that bay_1 contains $\overline{cv_{i-1}}$ as an edge (see Fig. 6). We show (in [6]) that $\overline{v_{i-1}p}$ must *appear in* $VD(bay(\overline{cd}))$ (which means that $\overline{v_{i-1}p}$ lies on some cell boundaries of $VD(bay(\overline{cd}))$) and the original problem of computing $VD(bay(\overline{cd}))$ on $bay(\overline{cd})$ and R can be broken into two subproblems of computing

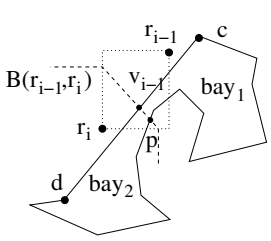

Figure 6 $B_M(r_{i-1}, r_i)$ intersects both $\overline{cd}$ (at v_{i-1}) and ∂ (at p). The line segment $\overline{v_{i-1}p}$ divides $bay(\overline{cd})$ into bay_1 and bay_2.

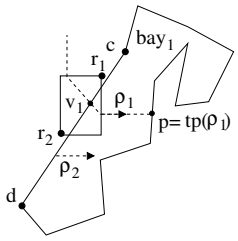

Figure 7 Illustrating an example of ρ_1 being horizontal.

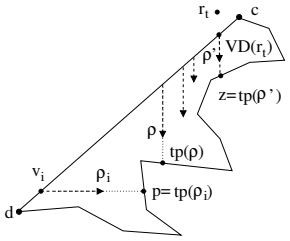

Figure 8 The target points of all rays in S (whose rays are all vertical) are before $p = tp(\rho_i)$. The ray ρ is at the top of S and ρ' is at the bottom of S.

$VD(bay_1)$ on bay_1 and $\{r_1, \ldots, r_{i-1}\}$ and computing $VD(bay_2)$ on bay_2 and $\{r_i, \ldots, r_k\}$. The above procedure is done in the preprocessing of the algorithm, where all (-1)-sloped ray shootings are solved in totally $O(n' + k)$ time. Thus, we only need to focus on each individual subproblem. W.l.o.g., we assume the original problem is one subproblem (i.e., no $B'_M(r_{i-1}, r_i)$ intersects ∂). We can show that each $B'_M(r_{i-1}, r_i)$ appears in $VD(bay(\overline{cd}))$ [6]. To compute $VD(bay(\overline{cd}))$, essentially we need to handle the interactions of all rays $\rho_1, \ldots, \rho_{k-1}$. Let $\Psi = \{\rho_1, \ldots, \rho_{k-1}\}$. Considering the roots in R incrementally is equivalent to considering the corresponding rays in Ψ incrementally. Specifically, our algorithm processes the rays in Ψ from ρ_1 to ρ_{k-1} incrementally. Recall that each $\rho_i \in \Psi$ is either vertically going south or horizontally going east. We use a stack S to store certain vertical rays and $S = \emptyset$ initially. As will be seen later, some rays in S may not be in Ψ. For a ray ρ with its origin in $bay(\overline{cd})$, the point on ∂ hit first by ρ is called the *target point* of ρ, denoted by $tp(\rho)$.

Our algorithm maintains a number of invariants, and the next paragraph lists a subset of them that are related to our discussion in this section. The complete list of invariants (as well as the argument why our algorithm implicitly maintains them) are in [6].

Let $\hat{\rho}$ be the next ray to be considered by the algorithm. Assume $S \neq \emptyset$. **Invariants:** (a) All rays in S are vertically going south. (b) The origins of all rays in S from top to bottom are ordered from southwest to northeast. (c) The origin of $\hat{\rho}$ is to the southwest of the origin of the ray at the top of S. (d) Suppose $\hat{\rho}$ is on a bisector $B(r_j, r_i)$ with $j < i$ and the ray at the top of S is on a bisector $B(r_t, r_{t'})$ with $t < t'$; then $j = t'$. (e) For each ray ρ'' in $S \cup \{\hat{\rho}\}$, suppose ρ'' lies on a bisector $B(r_{j'}, r_{i'})$ of two roots $r_{j'}$ and $r_{i'}$ with $j' < i'$; then the portion of the boundary of the Voronoi region $VD(r_{i'})$ (resp., $VD(r_{j'})$) from $v_{i'-1}$ (resp., $v_{j'}$) to the origin $or(\rho'')$ of ρ'' has already been computed. (f) The target points of all rays in S from bottom to top are ordered clockwise on ∂, i.e., from c to d (this property is called "target-sorted").

Consider the first ray ρ_1. If ρ_1 is vertical, we push it on S and continue to consider the ray $\rho_2 \in \Psi$. If ρ_1 is horizontal, we find the target point $p = tp(\rho_1)$ of ρ_1 (i.e., the first point on ∂ hit by ρ_1) by performing a horizontal ray shooting. Then, $B'_M(r_1, r_2)$ and $\overline{or(\rho_1)p}$ together partition $bay(\overline{cd})$ into two simple polygons, one of which contains $\overline{cv_1}$ as an edge and we denote it by bay_1 (see Fig. 7). We show that bay_1 is the Voronoi region of r_1, i.e., $VD(r_1) = bay_1$. We then continue to consider the ray $\rho_2 \in \Psi$.

Now consider a general step of the algorithm, which processes a ray $\rho_i \in \Psi$. If ρ_i is vertical, we simply push ρ_i on the top of the stack S and continue to consider the next ray $\rho_{i+1} \in \Psi$. Below, we discuss the case when ρ_i is horizontal. Let $p = tp(\rho_i)$ be the target point of ρ_i. If $S = \emptyset$, then as in the case of ρ_1, the Voronoi region $VD(r_i)$ is determined immediately (note that $\rho_i \in B(r_i, r_{i+1})$), and we continue to consider ρ_{i+1}. Below, we assume

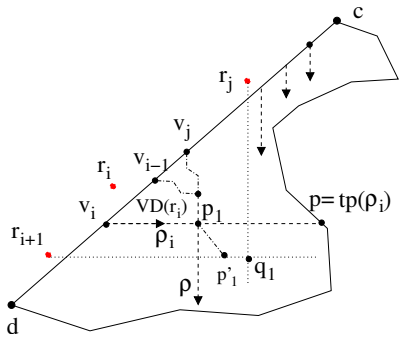

Figure 9 Illustrating an example that the ray ρ at the top of S intersects ρ_i (at p_1) before hitting ∂.

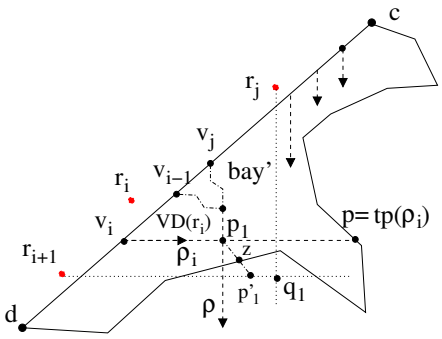

Figure 10 Illustrating an example that $B'_M(r_j, r_{i+1})$ $(= \overline{p_1 p'_1})$ intersects ∂ (first at z).

$S \neq \emptyset$. For any two points a and b on ∂ with a lying in the portion of ∂ from the vertex c clockwise to b, we say a is *before* b or b is *after* a. Suppose ρ is the ray on the top of S. By Invariant (b), ρ is the leftmost ray in S. If the target point $tp(\rho)$ is before p, then by Invariants (f), ρ_i does not intersect any vertical ray in S before they hit ∂ (see Fig. 8). We then perform a *splitting procedure* on the rays in S, as follows.

Let ρ' be the ray at the *bottom* of S and $z = tp(\rho')$ (see Fig. 8). Suppose ρ' is on a bisector, say $B(r_t, r_{t'})$ for some $t < t'$. By Invariant (e), the boundary portion of $VD(r_t)$ between v_t and the origin $or(\rho')$ has been computed. The concatenation of the segment $\overline{or(\rho')z}$ and the above boundary portion of $VD(r_t)$ splits the current region of $bay(\overline{cd})$ that needs to be further decomposed for computing $VD(bay(\overline{cd}))$ into two simple polygons. The one containing $\overline{v_{t-1}v_t}$ is the Voronoi region $VD(r_t)$. We then continue to process the second bottom ray in S in a similar fashion. This splitting procedure stops once all rays in S are processed. The target points of all rays in S are found by vertical ray shootings, which is done by a *scanning procedure* that basically scans a portion of ∂. Finally, we pop all rays out of S. We then continue to consider the next ray $\rho_{i+1} \in \Psi$.

If $tp(\rho)$ is after p, then ρ_i intersects ρ before they hit ∂ (e.g., see Fig. 9). Suppose ρ is on $B(r_j, r_{j'})$ with $j < j'$. Recall $\rho_i \in B(r_i, r_{i+1})$. By Invariant (d), $j' = i$, and the Voronoi region $VD(r_i)$ can be determined immediately [6]. Let p_1 be the intersection of ρ_i and ρ (see Fig. 9). Let q_1 be the intersection of the vertical line through r_j and the horizontal line through r_{i+1}. We show (in [6]) that q_1 must be to the southeast of p_1. The line of slope -1 through p_1 intersects the boundary of the rectangle $Rec(p_1, q_1)$ at two points: One is p_1 and denote the other one by p'_1 (see Fig. 9). We show that $\overline{p_1 p'_1} \subseteq B(r_j, r_{i+1})$ and $\overline{p_1 p'_1} \cap bay(\overline{cd})$ appears in $VD(bay(\overline{cd}))$. Depending on whether $\overline{p_1 p'_1}$ intersects ∂, there are two cases.

If $\overline{p_1 p'_1}$ intersects ∂, let z be the first intersection point (see Fig. 10). Similarly as before, the line segment $\overline{p_1 z}$ appears in $VD(bay(\overline{cd}))$ and partitions the current region of $bay(\overline{cd})$ that needs further decomposition for computing $VD(bay(\overline{cd}))$ into two simple polygons; one of them, say bay', contains the point p. Then, the Voronoi regions of the roots that define the rays in S form a decomposition of bay', and we use a procedure similar to the splitting procedure discussed earlier to compute this decomposition of bay', i.e., consider the rays in S from bottom to top. Finally, we pop all rays out of S, and continue with the next ray $\rho_{i+1} \in \Psi$.

If $\overline{p_1 p'_1}$ does not intersect ∂, then depending on whether p'_1 is on the bottom edge or the right edge of the rectangle $Rec(p_1, q_1)$, there are further two subcases. In either subcase, we first pop ρ out of S. If p'_1 is on the bottom edge, then let ρ_i^* be the vertical ray originating at p'_1 and going south (see Fig. 11). We call ρ_i^* the *termination vertical ray* of ρ_i. We push

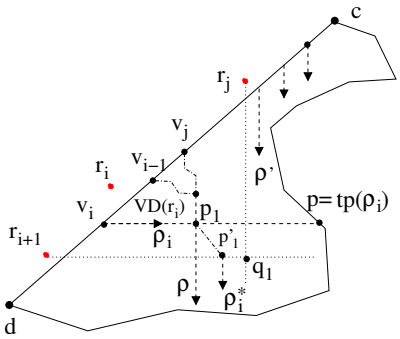

Figure 11 Illustrating an example that the point p'_1 $(= or(\rho_i^*))$ is on the bottom edge of $Rec(p_1, q_1)$.

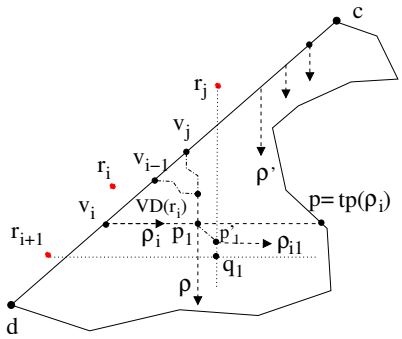

Figure 12 Illustrating an example that the point p'_1 $(= or(\rho_{i1}))$ is on the right edge of $Rec(p_1, q_1)$.

ρ_i^* on the top of S. We then continue to consider the next ray $\rho_{i+1} \in \Psi$. If p'_1 is on the right edge of $Rec(p_1, q_1)$, then let ρ_{i1} be the horizontal ray originating at p'_1 and going east (see Fig. 12). We call ρ_{i1} a *successor horizontal ray* of ρ_i. The ray ρ_{i1} (not ρ_{i+1}) is the next ray that will be considered by the algorithm. Note that $\rho_{i1} \notin \Psi$. We then continue with processing ρ_{i1}. We should mention that although our discussion above is on a ray ρ_i in Ψ, the processing for (the horizontal) ρ_{i1} is very similar to the case when $\rho_i \in \Psi$ is horizontal. In particular, there may also be a termination vertical ray or a successor horizontal ray generated after processing ρ_{i1}. Thus, the processing of a horizontal ray $\rho_i \in \Psi$ may lead to generating multiple successor horizontal rays but at most one termination vertical ray, i.e., a successor horizontal ray may generate another successor horizontal ray (e.g., see Fig. 13), but a termination vertical ray does not generate another ray.

We have discussed all possible cases for processing a ray. The algorithm finishes when the Voronoi regions for all roots in R are computed.

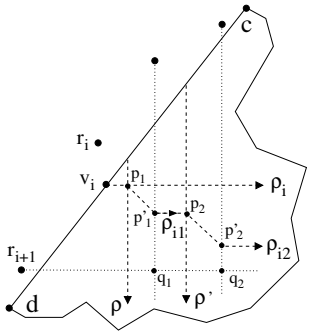

Figure 13 Illustrating the first two successor horizontal rays ρ_{i1} and ρ_{i2} of a horizontal ray $\rho_i \in \Psi$.

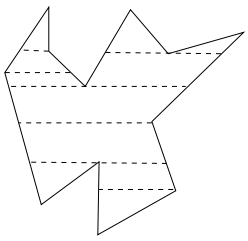

Figure 14 Illustrating the horizontal visibility map of a simple polygon.

The running time of the algorithm is $O(n' + m')$ (recall $m' = k + 1$ and n' is the number of vertices in $bay(\overline{cd})$). For the implementation, in the preprocessing we also compute a horizontal visibility map $HM(bay(\overline{cd}))$ (see Fig. 14) and a vertical visibility map $VM(bay(\overline{cd}))$ [2]. We do not overlap the two maps. In the main algorithm, we use $HM(bay(\overline{cd}))$ to guide the computation, i.e., we keep track of which trapezoid of $HM(bay(\overline{cd}))$ we are in during the algorithm. This allows each horizontal ray shooting to be performed in constant time. We also use $HM(bay(\overline{cd}))$ to compute the first intersection point of $\overline{p_1 p'_1}$ and ∂ (i.e., the point z in Fig. 10). To conduct the vertical ray shootings (i.e., for the rays in S), we utilize the vertical map $VM(bay(\overline{cd}))$ and a scanning procedure. Further, with Invariant (f), we

can show that the target points of the vertical rays in the entire algorithm that we need to compute are ordered on ∂ from c to d (i.e., the target-sorted property). In addition, we use a *reference point* p^* to help implement the vertical ray shootings. The reference point p^*, which is at c (resp., d) at the beginning (resp., end) of the algorithm, always moves (forward) on ∂ from c to d during the algorithm but it never moves backward. These components together perform all vertical ray shootings in totally $O(n' + m')$ time. Note that although there are known data structures for general ray shootings [3, 4, 12, 14], they are not efficient enough for our purpose. Also note that although the processing of a horizontal ray ρ_i in Ψ may produce multiple successor horizontal rays, we can show that the total number of horizontal rays in the entire algorithm is at most k and that of vertical rays is also at most k.

5 Expanding $SPM(\mathcal{M})$ into a Canal (a Sketch)

We sketch the idea of computing an SPM for a canal, say $canal(x, y)$. The details are in [6]. A main difference than the bay case is that a canal has two gates, say $\overline{xd}$ and $\overline{yz}$ (e.g., see Fig. 3). Let R_1 (resp., R_2) be the set of roots whose cells in $SPM(\mathcal{M})$ intersect $\overline{xd}$ (resp., $\overline{yz}$). Let $VD(canal(x, y), R_1)$ denote the weighted Voronoi diagram of $canal(x, y)$ with respect to R_1, i.e., we treat $canal(x, y)$ as a bay with the gate $\overline{xd}$. Define $VD(canal(x, y), R_2)$ similarly.

We first compute $VD(canal(x, y), R_1)$ and $VD(canal(x, y), R_2)$ by our algorithm for a bay in Section 4. We then find a "dividing curve" γ in $canal(x, y)$ that divides $canal(x, y)$ into two simple polygons C_1 and C_2, such that each point in C_1 (resp., C_2) has a shortest path from s via a root in R_1 (resp., R_2). We apply our algorithm for a bay on C_1 and R_1 to compute the weighted Voronoi diagram of C_1 with respect to R_1, i.e., $VD(C_1, R_1)$. We similarly compute $VD(C_2, R_2)$. It is easy to see that $SPM(canal(x, y))$ is a concatenation of $VD(C_1, R_1)$ and $VD(C_2, R_2)$. It remains to compute the dividing curve γ.

To compute γ, we first determine a point $p^* \in \gamma$ (e.g., with the help of the corridor path). Then, we trace γ out from p^* by traversing the cells of $VD(canal(x, y), R_1)$ and $VD(canal(x, y), R_2)$, which is similar to the merge procedure of the divide-and-conquer Voronoi diagram algorithm [20].

References

1 R. Bar-Yehuda and B. Chazelle. Triangulating disjoint Jordan chains. *International Journal of Computational Geometry and Applications*, 4(4):475–481, 1994.

2 B. Chazelle. Triangulating a simple polygon in linear time. *Discrete and Computational Geometry*, 6:485–524, 1991.

3 B. Chazelle, H. Edelsbrunner, M. Grigni, L. Gribas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12(1):54–68, 1994.

4 B. Chazelle and L. Guibas. Visibility and intersection problems in plane geometry. *Discrete and Computational Geometry*, 4:551–589, 1989.

5 D.Z. Chen and H. Wang. A nearly optimal algorithm for finding L_1 shortest paths among polygonal obstacles in the plane. In *Proc. of the 19th European Symposium on Algorithms*, pages 481–492, 2011.

6 D.Z. Chen and H. Wang. Computing L_1 shortest paths among polygonal obstacles in the plane. arXiv:1202.5715v1, 2012.

7 D.Z. Chen and H. Wang. Computing the visibility polygon of an island in a polygonal domain. In *Proc. of the 39th International Colloquium on Automata, Languages and Programming*, pages 218–229, 2012.

8 K. Clarkson, S. Kapoor, and P. Vaidya. Rectilinear shortest paths through polygonal obstacles in $O(n \log^2 n)$ time. In *Proc. of the 3rd Annual Symposium on Computational Geometry*, pages 251–257, 1987.

9 P.J. de Rezende, D.T. Lee, and Y.F. Wu. Rectilinear shortest paths in the presence of rectangular barriers. *Discrete and Computational Geometry*, 4:41–53, 1989.

10 H. Edelsbrunner, L. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal on Computing*, 15(2):317–340, 1986.

11 S. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987.

12 L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R.E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1-4):209–233, 1987.

13 J. Hershberger and J. Snoeyink. Computing minimum length paths of a given homotopy class. *Computational Geometry: Theory and Applications*, 4(2):63–97, 1994.

14 J. Hershberger and S. Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *Journal of Algorithms*, 18(3):403–431, 1995.

15 R. Inkulu and S. Kapoor. Planar rectilinear shortest path computation using corridors. *Computational Geometry: Theory and Applications*, 42(9):873–884, 2009.

16 S. Kapoor, S.N. Maheshwari, and J.S.B. Mitchell. An efficient algorithm for Euclidean shortest paths among polygonal obstacles in the plane. *Discrete and Computational Geometry*, 18(4):377–383, 1997.

17 D. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983.

18 J.S.B. Mitchell. An optimal algorithm for shortest rectilinear paths among obstacles. Abstracts of the *1st Canadian Conference on Computational Geometry*, 1989.

19 J.S.B. Mitchell. L_1 shortest paths among polygonal obstacles in the plane. *Algorithmica*, 8(1):55–88, 1992.

20 M.I. Shamos and D. Hoey. Closest-point problems. In *Proc. of the 16th Annual Symposium on Foundations of Computer Science*, pages 151–162, 1975.

21 P. Widmayer. On graphs preserving rectilinear shortest paths in the presence of obstacles. *Annals of Operations Research*, 33(7):557–575, 1991.