Farthest-Point Voronoi Diagrams in the Presence of Rectangular Obstacles

Mincheol Kim
Department of Computer Science and Engineering,
Pohang University of Science and Technology, South Korea

Chanyang Seo
Graduate School of Artificial Intelligence,
Pohang University of Science and Technology, South Korea

Taehoon Ahn
Department of Computer Science and Engineering,
Pohang University of Science and Technology, South Korea

Hee-Kap Ahn
Graduate School of Artificial Intelligence, Department of Computer Science and Engineering,
Pohang University of Science and Technology, South Korea

Abstract

We present an algorithm to compute the geodesic L_1 farthest-point Voronoi diagram of m point sites in the presence of n rectangular obstacles in the plane. It takes $O(nm + n \log n + m \log m)$ construction time using $O(nm)$ space. This is the first optimal algorithm for constructing the farthest-point Voronoi diagram in the presence of obstacles. We can construct a data structure in the same construction time and space that answers a farthest-neighbor query in $O(\log(n + m))$ time.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Geodesic distance, L_1 metric, farthest-point Voronoi diagram

Digital Object Identifier 10.4230/LIPIcs.SoCG.2022.51

Related Version *Full Version*: `http://arxiv.org/abs/2203.03198`

Funding This research was partly supported by the Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No. 2017-0-00905, Software Star Lab (Optimal Data Structure and Algorithmic Applications in Dynamic Geometric Environment)) and (No. 2019-0-01906, Artificial Intelligence Graduate School Program(POSTECH)).

1 Introduction

A Voronoi diagram of a set of sites is a subdivision of the space under consideration into subspaces by assigning points to sites with respect to a certain proximity. Typical Voronoi assignment models are the nearest-point model and the farthest-point model where every point is assigned to its nearest site and its farthest site, respectively. There are results for computing Voronoi diagrams in the plane [1, 13, 14, 24], under different metrics [9, 17, 18, 23], or for various types of sites [2, 8, 22].

For m point sites in the plane, the nearest-point and farthest-point Voronoi diagrams of the sites can be constructed in $O(m \log m)$ time [14, 24]. When the sites are contained in a simple polygon with no holes, the distance between any two points in the polygon, called the *geodesic distance*, is measured as the length of the shortest path contained in the polygon and connecting the points (called the *geodesic path*). There has been a fair amount of work computing the geodesic nearest-point and farthest-point Voronoi diagrams of m point sites in a simple n-gon [3, 4, 20, 21] to achieve the lower bound $\Omega(n + m \log m)$ [3]. Recently, optimal algorithms of $O(n + m \log m)$ time were given for the geodesic nearest-point Voronoi diagram [19] and for the geodesic farthest-point Voronoi diagram [25].

The problem of computing Voronoi diagrams is more challenging in the presence of obstacles. Each obstacle plays as a hole and there can be two or more geodesic paths connecting two points avoiding those holes. The geodesic nearest-point Voronoi diagram of m point sites can be computed in $O(m \log m + k \log k)$ time by applying the continuous Dijkstra paradigm [16], where k is the number of total vertices of obstacles. However, no optimal algorithm is known for the farthest-point Voronoi diagram in the presence of obstacles in the plane, even when the obstacles are of elementary shapes such as axis-aligned line segments and rectangles. The best result of the geodesic farthest-point Voronoi diagram known so far takes $O(mk \log^2(m + k) \log k)$ time by Bae and Chwa [5]. They also showed that the total complexity of the geodesic farthest-point Voronoi diagram is $\Theta(mk)$.

In the presence of n rectangular obstacles under L_1 metric, there are some work for farthest-neighbor queries. Ben-Moshe et al. [7] presented a data structure with $O(nm \log(n + m))$ construction time and $O(nm)$ space for m point sites that supports farthest point queries in $O(\log(n + m))$ time. They also showed that the L_1 geodesic farthest-point Voronoi diagram has complexity $\Theta(nm)$, but without presenting any algorithm for computing the diagram. Later Ben-Moshe et al. [6] gave a tradeoff between the query time and the preprocessing/space such that a data structure of size $O((n+m)^{1.5})$ can be constructed in $O((n+m)^{1.5} \log^2(n+m))$ to support farthest point queries in $O((n + m)^{0.5} \log(n + m))$ time.

The geodesic center of a set of objects in a polygonal domain is the set of points in the domain that minimize the maximum geodesic distance from input objects. Thus, it can be obtained once the geodesic farthest-point Voronoi diagram of the objects is constructed. For m points in the presence of n axis-aligned rectangular obstacles in the plane, Choi et al. [10] showed that the geodesic center of the points under the L_1 metric consists of $\Theta(nm)$ connected regions and they gave an $O(n^2m)$-time algorithm to compute the geodesic center. Later, Ben-Moshe et al. [7] gave an $O(nm \log(n + m))$-time algorithm for the problem.

Our Result. In this paper, we present an algorithm that computes the geodesic L_1 farthest-point Voronoi diagram of m points in the presence of n rectangular obstacles in the plane in $O(nm + n \log n + m \log m)$ time using $O(nm)$ space. The running time and space complexity of our algorithm match the time and space bounds of the Voronoi diagram. Thus, it is the first optimal algorithm for computing the geodesic farthest-point Voronoi diagram in the presence of obstacles.

To do this, we construct a data structure for L_1 farthest-neighbor queries in $O(nm + n \log n + m \log m)$ time using $O(nm)$ space. This improves upon the results by Ben-Moshe et al. [7], and the construction time and space are the best among the data structures supporting $O(\log(n+m))$ query time for L_1 farthest neighbors. Then we present an optimal algorithm to compute the explicit geodesic L_1 farthest-point Voronoi diagram in $O(nm+n \log n+m \log m)$ time using $O(nm)$ space, which matches the time and space lower bounds of the diagram.

As a byproduct, we compute the geodesic center under the L_1 metric in $O(nm + n \log n + m \log m)$ time. This result improves upon the algorithm by Ben-Moshe et al. [7].

Outline. First, we construct four farthest-point maps, one for each of the four axis directions, either the x- or y-axis, and either positive or negative. In the course, we construct a data structure for L_1 farthest-neighbor queries in $O(nm + n \log n + m \log m)$ time using $O(nm)$ space. For each axis direction, we apply the plane sweep technique with a line orthogonal to the direction and moving along the direction. During the sweep, we maintain the status of the sweep line in a balanced binary search tree and its associated structures while handling events induced by the point sites and the sides of rectangles parallel to the sweep line. There are m events induced by point sites and $O(n)$ events induced by rectangles. After sorting the events in $O(n \log n + m \log m)$ time, we show that we can handle all events induced by point sites in $O(nm)$ time. Additionally, we show that each event induced by a rectangle can be handled in $O(m + \log n)$ time. By the plane sweep, we construct a data structure consisting of $O(n + m)$ line segments parallel to the sweep line and $O(nm)$ points in $O(nm + n \log n + m \log m)$ time in total. Given a query, it uses axis-aligned ray shooting queries on the data structure to find the farthest site from the query. The four farthest-point maps are planar subdivisions, and they can be constructed during the plane sweep in the same time and space.

With the four farthest-point maps and the data structure for farthest-neighbor queries, we construct the geodesic L_1 farthest-point Voronoi diagram explicitly. First, we decompose the plane, excluding the holes, into rectangular faces using vertical line segments, each extended from a vertical side of a hole. Then, we partition each face in the decomposition into zones such that the farthest-point Voronoi diagram restricted to a zone coincides with the corresponding region of a farthest-point map. This partition is done by using the boundary between two farthest-point maps, which can be computed by traversing the cells in the two maps in which the boundary lies. Finally, we glue the corresponding regions along the boundaries of zones, and then glue all adjacent faces along their boundaries to obtain the geodesic L_1 farthest-point Voronoi diagram. We show that this can be done in $O(nm + n \log n + m \log m)$ time in total.

For the centers of m points in the presence of n axis-aligned rectangles in the plane, we can find them from the farthest-point Voronoi diagram in time linear to the complexity of the diagram.

2 Preliminaries

Let R be a set of n open disjoint rectangles and S be a set of m point sites lying in the *free space* $\mathsf{F} = \mathbb{R}^2 - \bigcup_{R \in \mathsf{R}} R$. We consider the L_1 metric. For ease of description, we omit L_1. We use $x(p)$ and $y(p)$ to denote the x-coordinate and y-coordinate of a point p, respectively. For two points p and q in F, we use pq to denote the line segment connecting them. Whenever we say a path connecting two points in F, it is a path contained in F. There can be more than one geodesic path connecting two points p and q avoiding the holes. We use $\pi(p, q)$ to denote a fixed geodesic path connecting p and q, and use $d(p, q)$ to denote the geodesic distance between p and q, which is the length of $\pi(p, q)$.

We make a general position assumption that no point in F is equidistant from four or more distinct sites. We use $f(p)$ to denote the set of sites of S that are farthest from a point $p \in \mathsf{F}$ under the geodesic distance, that is, a site s is in $f(p)$ if and only if $d(s, p) \geq d(s', p)$ for all $s' \in \mathsf{S}$. If there is only one farthest site, we use $f(p)$ to denote the site.

A horizontal line segment ℓ can be represented by the two x-coordinates $x_1(\ell)$ and $x_2(\ell)$ of its endpoints ($x_1(\ell) < x_2(\ell)$) and the y-coordinate $y(\ell)$ of them. For an axis-aligned rectangle R, let $x_1(R)$ and $x_2(R)$ denote the x-coordinates of the left and right sides of R.

A path is *x-monotone* if and only if the intersection of the path with any line perpendicular to the x-axis is connected. Likewise, a path is *y-monotone* if and only if the intersection of the path with any line perpendicular to the y-axis is connected. A path is *xy-monotone* if and only if the path is x-monotone and y-monotone. Observe that if a path connecting two points is xy-monotone, it is a geodesic path connecting the points.

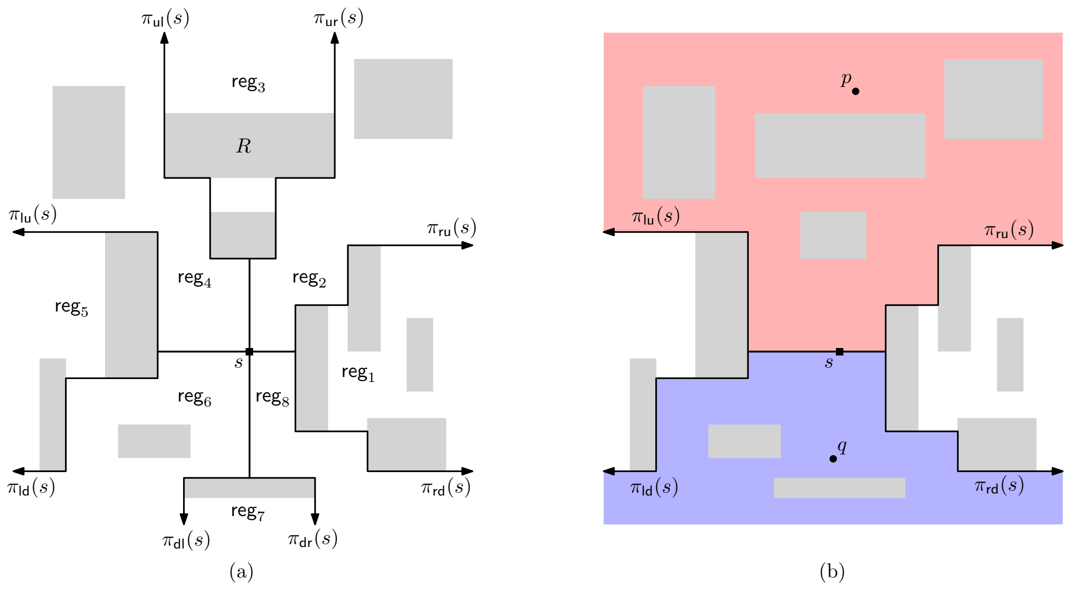

Figure 1 Gray rectangles are holes. (a) The eight paths partition F into eight regions $\mathsf{reg}_1, \ldots, \mathsf{reg}_8$. Region reg_3 consists of two regions separated by a rectangle R. (b) Every geodesic path from s to p is y^+-monotone and p is y^+-reachable from s. Every geodesic path from s to q is y^--monotone and q is y^--reachable from s.

2.1 Eight Monotone Paths from a Point

Choi and Yap [11] gave a way of partitioning the plane with rectangular holes into eight regions using eight xy-monotone paths from a point. We use their method to partition F as follows. Consider a horizontal ray emanating from a point $s = p_1 \in \mathsf{F}$ going rightwards. The ray stops when it hits a rectangle $R \in \mathsf{R}$ at a point p_1'. Let p_2 be the top-left corner of R. We repeat this process by taking a horizontal ray from p_2 going rightwards until it hits a rectangle, and so on. Then we obtain an xy-monotone path $\pi_{\mathsf{ru}}(s) = p_1 p_1' p_2 p_2' \ldots$ from s that alternates going *rightwards* and going *upwards*.

By choosing two directions, one going either rightwards or leftwards horizontally, and one going either upwards or downwards vertically, and ordering the chosen directions, we define eight rectilinear xy-monotone paths with directions: rightwards-upwards (ru), upwards-rightwards (ur), upwards-leftwards (ul), leftwards-upwards (lu), leftwards-downwards (ld), downwards-leftwards (dl), downwards-rightwards (dr), and rightwards-downwards (rd). Let $\pi_\delta(s)$ denote one of the eight paths corresponding to the direction δ in $\{\mathsf{ru}, \mathsf{ur}, \mathsf{ul}, \mathsf{lu}, \mathsf{ld}, \mathsf{dl}, \mathsf{dr}, \mathsf{rd}\}$.

Some of the eight paths $\pi_\delta(s)$ may overlap in the beginning from s but they do not cross each other. The paths partition F into eight regions $\mathsf{reg}_1, \ldots, \mathsf{reg}_8$ with the indices sorted around s in a counterclockwise order such that reg_1 denotes the region lying to the right of s, below $\pi_{\mathsf{ru}}(s)$ and above $\pi_{\mathsf{rd}}(s)$. Observe that reg_i is not necessarily connected. See Figure 1(a) for an illustration.

▶ **Lemma 1** ([11, 12]). *Every geodesic path connecting two points is either x-, y-, or xy-monotone. For a point $s \in \mathsf{F}$, following three statements hold.*

- *If $p \in \mathsf{reg}_1 \cup \mathsf{reg}_5$, every geodesic path from s to p is x-monotone but not y-monotone.*
- *If $p \in \mathsf{reg}_3 \cup \mathsf{reg}_7$, every geodesic path from s to p is y-monotone but not x-monotone.*
- *If $p \in \mathsf{reg}_2 \cup \mathsf{reg}_4 \cup \mathsf{reg}_6 \cup \mathsf{reg}_8 \cup \Pi(s)$, every geodesic path from s to p is xy-monotone, where $\Pi(s)$ is the union of the eight paths $\pi_\delta(s)$.*

Based on Lemma 1, we define a few more terms. For any point p in $\mathsf{reg}_2 \cup \mathsf{reg}_3 \cup \mathsf{reg}_4$ (and the boundaries of the regions), we say p is *y^+-reachable* from s, and every geodesic path from s to p is *y^+-monotone.* Any point $q \in \mathsf{reg}_6 \cup \mathsf{reg}_7 \cup \mathsf{reg}_8$ (and the boundaries of the regions) is *y^--reachable* from s, and every geodesic path from s to q is *y^--monotone.* See Figure 1(b). Similarly, any point $p \in \mathsf{reg}_1 \cup \mathsf{reg}_2 \cup \mathsf{reg}_8$ (and the boundaries of the regions) is *x^+-reachable* from s, and every geodesic path from s to p is *x^+-monotone.* Any point $q \in \mathsf{reg}_4 \cup \mathsf{reg}_5 \cup \mathsf{reg}_6$ (and the boundaries of the regions) is *x^--reachable* from s, and every geodesic path from s to q is *x^--monotone.*

3 Farthest-point Maps

Based on Lemma 1 and the four directions of monotone paths in the previous section, we define four *farthest-point maps.* A farthest-point map $\mathsf{M}_{y^+} = \mathsf{M}_{y^+}(\mathsf{S})$ of S in F corresponding to the positive y-direction is a planar subdivision of F into cells. For a point $p \in \mathsf{F}$, a site $s \in \mathsf{S}$ is a farthest site of p in M_{y^+} if $d(p,s) \geq d(p,s')$ for every site $s' \in \mathsf{S}$ from which p is y^+-reachable. If p is y^+-reachable from no site in S, p has no farthest site in M_{y^+}. Thus, a cell of M_{y^+} is defined on $\mathsf{F} \setminus C_\emptyset$, where $C_\emptyset$ denotes the set of points of F that are y^+-reachable from no site in S. A site s corresponds to one or more cells in M_{y^+} with the property that a point $p \in \mathsf{F} \setminus C_\emptyset$ lies in a cell of s if and only if $d(p,s) > d(p,s')$ for every $s' \in \mathsf{S} \setminus \{s\}$ from which p is y^+-reachable.

We define M_{y^-}, M_{x^+} and M_{x^-} analogously with respect to their corresponding directions. Since the four maps have the same structural and combinatorial properties with respect to their corresponding directions, we describe only M_{y^+} in the following. Let B be an axis-aligned rectangular box such that S, R, and all vertices of the four farthest-point maps are contained in the interior of B. We focus on $\mathsf{F} \cap \mathsf{B}$ only, and use F as $\mathsf{F} \cap \mathsf{B}$.

In the following, we analyze the edges of M_{y^+} using the bisectors of pairs of sites. Let $F(s,s')$ denote a set of points of F that are y^+-reachable from two sites s and s'. To be specific, $F(s,s')$ is an intersection of two regions, one lying above $\pi_{\mathsf{lu}}(s)$ and $\pi_{\mathsf{ru}}(s)$ and the other lying above $\pi_{\mathsf{lu}}(s')$ and $\pi_{\mathsf{ru}}(s')$. Thus, the boundary of $F(s,s')$ coincides with the upper envelope of $\pi_{\mathsf{lu}}(s)$, $\pi_{\mathsf{ru}}(s)$, $\pi_{\mathsf{lu}}(s')$ and $\pi_{\mathsf{ru}}(s')$. We use $F(s,s)$ to denote the set of points that are y^+-reachable from a site s.

For any two distinct sites $s, s' \in \mathsf{S}$, their *bisector* consists of all points $x \in \mathsf{F}$ satisfying $\{x \mid d(x,s) = d(x,s')\}$. Observe that the bisector may contain a two-dimensional region. We use $b(s,s')$ to denote the line segments and the boundary of the two-dimensional region in the bisector of s and s'.

A proof of the following lemma is given in the full version.

▶ **Lemma 2.** *For any two sites s and s', $b(s,s') \cap F(s,s')$ consists of axis-aligned segments.*

Let $f_\delta(p)$ denote the set of farthest sites from a point $p \in \mathsf{F}$ among the sites from which p is δ-reachable for $\delta \in \{y^+, y^-, x^+, x^-\}$. For each horizontal segment of $\pi_{\mathsf{lu}}(s) \cup \pi_{\mathsf{ru}}(s)$, we call the portion h of the segment such that $f_{y^+}(p) = \{s\}$ for any point $p \in h$, a *b-edge*. Observe that no point p' with $x_1(h) \leq x(p') \leq x_2(h)$ and $y(p') = y(h) - \varepsilon$ for any $\varepsilon > 0$ is y^+-reachable from s. Thus, a b-edge is also an edge of M_{y^+}. Since every edge of M_{y^+} is part of a bisector of two sites in S or a b-edge, it is either horizontal or vertical. See Figure 2(a).

▶ **Corollary 3.** *Every edge of M_{y^+} is an axis-aligned line segment.*

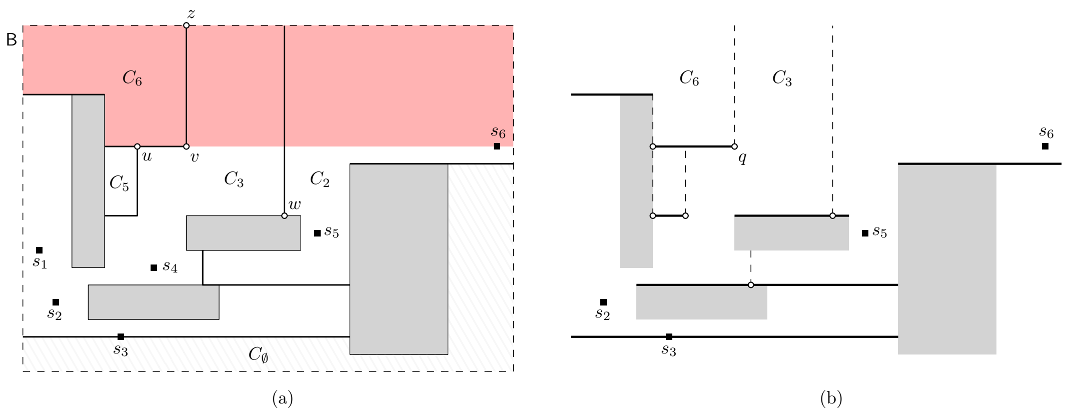

Figure 2 (a) M_{y^+} for $\mathsf{S} = \{s_1, \dots, s_6\}$ restricted to a box B with four rectangular holes (gray). s_i has a corresponding cell C_i for $i = 2, 3, 5, 6$ while s_1 and s_4 have no cell. A vertical edge vz is from $b(s_3, s_6)$ in the (red) region $F(s_3, s_6)$. A horizontal edge uv is not part of $b(s_3, s_6)$ but it is part of a b-edge as no point lying below uv is y^+-reachable from s_6. (b) Illustration of Q_{y^+} corresponding to M_{y^+}. At the boundary point q, $d(q, s_3) = d(q, s_6)$.

For sites contained in a simple polygon, Aronov et al. [4] gave a lemma, called *Ordering Lemma*, that the order of sites along their convex hull is the same as the order of their Voronoi cells along the boundary of a simple polygon. We give a lemma on the order of sites in the presence of rectangular obstacles. We use it in analyzing the maps and Voronoi diagrams. A proof of the following lemma is given in the full version.

▶ **Lemma 4.** *Let pq be a horizontal segment contained in $\mathsf{F} \setminus C_\emptyset$ with $x(p) < x(q)$. For any two sites $f_p \in f(p)$ and $f_q \in f(q)$ such that p and q are y^+-reachable from both f_p and f_q, if $f_p \notin f(q)$ or $f_q \notin f(p)$, $x(f_p) > x(f_q)$.*

Since there are at most m sites, we obtain the following corollary from Lemma 4.

▶ **Corollary 5.** *Any horizontal line segment contained in F intersects at most m cells in M_{y^+}.*

Using Corollary 3 and 5, we analyze the complexity of M_{y^+} as follows. Note that each lower endpoint of a vertical edge of M_{y^+} appears on a horizontal line segment passing through a site or the top side of a rectangle. By Corollary 5, the maximal horizontal segment through the top side of a rectangle in R and contained in F intersects $O(m)$ vertical edges of M_{y^+}. Moreover, the maximal horizontal line segment through a site s and contained in F intersects $O(1)$ lower endpoints of vertical edges on the boundary of the cell of s. Since there are n rectangles in R and m sites in S, M_{y^+} has $O(nm + m) = O(nm)$ vertical edges. Every horizontal edge of M_{y^+} is a segment of a bisector or a b-edge, and it is incident to a side of a rectangle or another vertical edge. Since there are $O(n)$ rectangle sides, and $O(1)$ horizontal edges of M_{y^+} that are incident to a vertical edge, M_{y^+} has $O(n + nm) = O(nm)$ horizontal edges. Thus, M_{y^+} has complexity $O(nm)$.

Now we show that every farthest site $s \in f(p)$ of a point p in F is one of the farthest sites of p in the four farthest-point maps. By the definition of the farthest-point maps, p is contained in a cell of M_{y^+}, M_{y^-}, M_{x^+} or M_{x^-}. Since every geodesic path connecting two points is either y^+-, y^--, x^+-, or x^--monotone by Lemma 1, $s \in f(p)$ is one of the farthest sites of p in the four farthest-point maps. If p is contained in cells of two or more maps, we compare their distances to the farthest sites defining the cells and take the ones with the largest distance as the farthest sites of p. Thus, once the four farthest-point maps are constructed, the farthest sites of a query point can be computed from the map.

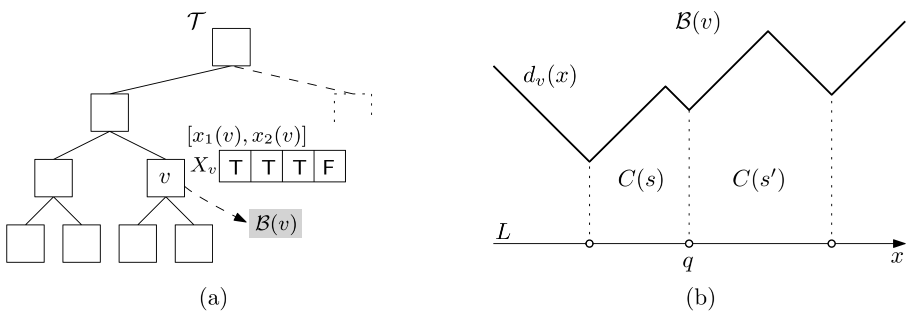

Figure 3 (a) Illustration of a balanced binary search tree $\mathcal{T}$. A node v in $\mathcal{T}$ has domain $[x_1(v), x_2(v)]$, array X_v, and a pointer to $\mathcal{B}(v)$. (b) Illustration of $\mathcal{B}(v)$ and $d_v(x)$.

4 Data Structure for Farthest-neighbor Queries

We present an algorithm that constructs a data structure for farthest site queries. We denote m point sites of S by $s_1, \ldots, s_m$ such that $x(s_1) \leq \cdots \leq x(s_m)$, and n rectangular obstacles of R by $R_1, \ldots, R_n$. The data structure consists of four parts, each for one axis direction. Since the four parts can be constructed in the same way with respect to their directions, we focus on the part corresponding to the positive y-direction, and thus the structure corresponds to M_{y^+}. We use Q_{y^+} to denote the query data structure.

By Corollary 3, we can find the farthest site of a query point using a vertical ray shooting query to the horizontal edges of M_{y^+} and a binary search on the lower endpoints of vertical edges of M_{y^+} lying on the horizontal edges of M_{y^+}. Thus, we construct Q_{y^+} such that it consists of the horizontal edges of M_{y^+} and the endpoints of vertical edges of M_{y^+} lying on the horizontal edges of M_{y^+}.

A point q lying on a horizontal segment h of Q_{y^+} is the lower endpoint of a vertical edge of M_{y^+} if and only if there are two points $q_1 = (x(q) - \varepsilon, y(q))$ and $q_2 = (x(q) + \varepsilon, y(q))$ for sufficiently small $\varepsilon > 0$ satisfying $f_{y^+}(q_1) \cup f_{y^+}(q_2) = f_{y^+}(q)$ and $f_{y^+}(q_1) \neq f_{y^+}(q_2)$. We call each lower endpoint of vertical edges lying on h a *boundary point* on h. See Figure 2(b).

We use a plane sweep algorithm with a horizontal sweep line L to construct the horizontal line segments in Q_{y^+}. Note that $\mathsf{F} \cap L$ consists of disjoint horizontal segments along L. The status of L is the sequence of segments in $\mathsf{F} \cap L$ along L. The status changes while L moves upwards over the plane, but not continuously. Each update of the status occurs at a particular y-coordinate, which we call an *event*. To do such updates efficiently, we maintain three data structures for L: a *balanced binary search tree* $\mathcal{T}$ representing the status, a *boundary list* $\mathcal{B}$, and a list $\mathcal{D}$ of distance functions. The structures $\mathcal{B}$ and $\mathcal{D}$ are associated structures of $\mathcal{T}$.

We store the segments of $\mathsf{F} \cap L$ in a balanced binary search tree $\mathcal{T}$ in increasing order of x-coordinate of their left endpoints. Each node v of $\mathcal{T}$ corresponds to a horizontal line segment h_v of $\mathsf{F} \cap L$. We store $x_1(h_v)$ and $x_2(h_v)$, and an array X_v of m Boolean variables at v. We set $X_v[i] = \mathtt{T}$ if a point on h_v is y^+-reachable from s_i for $i = 1 \ldots, m$. Otherwise, we set $X_v[i] = \mathtt{F}$. The range of v is $[x_1(v), x_2(v)]$ for $x_1(v) = x_1(h_v)$ and $x_2(v) = x_2(h_v)$. There are at most $n + 1$ nodes in $\mathcal{T}$, and each node maintains an array of size $O(m)$, so $\mathcal{T}$ itself uses $O(nm)$ space in total. See Figure 3(a).

The list $\mathcal{B}$ consists of boundary lists $\mathcal{B}(v)$ for nodes v of $\mathcal{T}$. Each node v of $\mathcal{T}$ has a pointer to its boundary list $\mathcal{B}(v)$, which is a doubly-linked list of the boundary points (including the endpoints of h_v) lying on h_v. Each boundary point in $\mathcal{B}$ is the intersection of L and a vertical edge of M_{y^+}, so there are $O(nm)$ boundary points in $\mathcal{B}$.

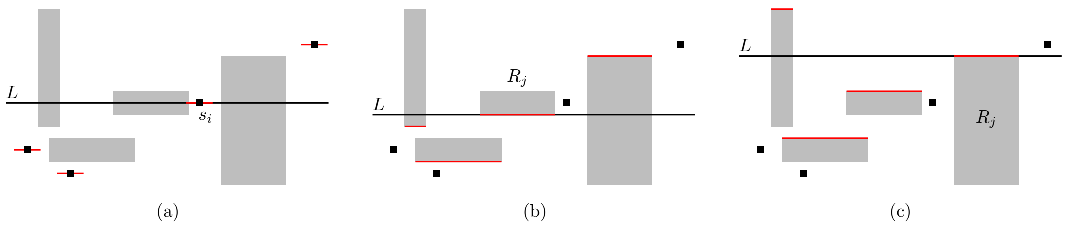

Figure 4 Three types of events. (a) site events. (b) bottom-side events. (c) top-side events.

Let $d_\delta(p) = d(s,p)$ for a site $s \in f_\delta(p)$ if $f_\delta(p) \neq \emptyset$, or $d_\delta(p) = -\infty$ for $\delta \in \{y^+, y^-, x^+, x^-\}$. The list $\mathcal{D}$ consists of distance functions d_v for nodes v of $\mathcal{T}$. Let $p(r)$ denote a point on L with $x(p(r)) = r$ for a real number r. Each node v of $\mathcal{T}$ has a pointer to its distance function $d_v(x) = d_{y^+}(p(x))$ for x in the range $[x_1(v), x_2(v)]$ of v. It is a piecewise linear function with pieces (segments) of slopes 1 or -1. See Figure 3(b).

There are three types of events: (1) a site event, (2) a bottom-side event, and (3) a top-side event. A site event occurs when L encounters a site in S. A bottom-side event occurs when L encounters the bottom side of a rectangle in R. A top-side event occurs when L encounters the top side of a rectangle in R. Thus, there are m site events, n bottom-side events, and n top-side events. See Figure 4.

We maintain and update $\mathcal{T}$, $\mathcal{B}$ and $\mathcal{D}$ during the plane sweep for those events. To handle events, we first sort the events in y-coordinate order, which takes $O((n+m)\log(n+m)) = O(n\log n + m\log m)$ time. We update $d_v(x)$ only at those events and keep it unchanged between two consecutive events. To reflect the distances from sites to $p(x) \in h_v$ correctly, we assign an additive weight to $d_v(x)$, which is the difference in the y-coordinates between the current event and the last event at which $d_v(x)$ is updated.

Initially, when L is at the bottom side of B, $\mathcal{T}$ consists of one node v with $x_1(v) = x_1(\mathsf{B})$, $x_2(v) = x_2(\mathsf{B})$, and $X_v[i] = \mathtt{F}$ for all $i \in \{1, \dots, m\}$. $\mathcal{B}(v)$ has no boundary point and $d_v(x) = -\infty$ for all x, since no points on L is y^+-reachable from any sites.

4.1 Handling a site event

When L encounters a site $s_i \in \mathsf{S}$, we find the node $v \in \mathcal{T}$ such that $x_1(v) \leq x(s_i) \leq x_2(v)$. Every point on h_v is y^+-reachable from s_i, so we set $X_v[i] = \mathtt{T}$. We can find v in $O(\log n)$ time, and set $X_v[i] = \mathtt{T}$ in constant time. Thus, it takes $O(\log n)$ time to update $\mathcal{T}$.

For any point $p(x) \in h_v$, $d(s_i, p(x)) = |x - x(s_i)|$. By Lemma 4, there is at most one maximal interval $I \subset [x_1(v), x_2(v)]$ such that $d_v(x) < d(s_i, p(x))$ for every $x \in I$. Moreover, I is bounded from left by $x_1(v)$ or from right by $x_2(v)$ because $d_v(x)$ is continuous and consists of pieces (segments) of slopes 1 or -1, and $d(s_i, p(x)) = |x - x(s_i)|$. We find the boundary point $p(x^*) \in h_v$ induced by s_i such that $d_v(x^*) = d(s_i, p(x^*))$. If I is bounded from left, we update $d_v(x)$ to $d_v(x) = d(s_i, p(x))$ for $x \leq x^*$. If I is bounded from right, we update $d_v(x)$ to $d_v(x) = d(s_i, p(x))$ for $x \geq x^*$.

If there is no such point $p(x^*)$, either $d_v(x) < d(s_i, p(x))$ or $d_v(x) > d(s_i, p(x))$ for all x with $x_1(v) \leq x \leq x_2(v)$. If $d_v(x) < d(s_i, p(x))$, we update $d_v(x)$ to $d_v(x) = d(s_i, p(x))$ for $x_1(v) \leq x \leq x_2(v)$. If $d_v(x) > d(s_i, p(x))$, we do not update $d_v(x)$.

We update $\mathcal{B}(v)$ by removing all the boundary points of $\mathcal{B}(v)$ lying in the interior of I in time linear to the number of the boundary points, and then inserting $p(x^*)$ into $\mathcal{B}(v)$.

Since there are m site events, it takes $O(m \log n)$ time in total to update $\mathcal{T}$. The total time to remove the boundary points is linear to the total number of boundary points in Q_{y^+}, which is $O(nm)$.

▶ **Lemma 6.** *We can handle all site events in $O(nm)$ time using $O(nm)$ space.*

4.2 Handling a bottom-side event

When L encounters the bottom side of a rectangle $R \in \mathsf{R}$, the line segment of $\mathsf{F} \cap L$ incident to the bottom side is replaced by two line segments by the event. See Figure 4(b). Thus, we update $\mathcal{T}$ by finding the node $v \in \mathcal{T}$ with $x_1(v) \leq x_1(R) < x_2(R) \leq x_2(v)$, removing v from $\mathcal{T}$, and then inserting two new nodes u and w into $\mathcal{T}$. We set $(x_1(u), x_2(u)) = (x_1(v), x_1(R))$, $(x_1(w), x_2(w)) = (x_2(R), x_2(v))$, $X_u = X_v$, and $X_w = X_v$. This takes $O(\log n)$ time since $\mathcal{T}$ is a balanced binary search tree. It takes $O(m)$ time to copy the Boolean values of X_v to X_u and X_w, and to remove X_v. Thus, it takes $O(m + \log n)$ time to update $\mathcal{T}$.

We update $\mathcal{B}$ by inserting two lists $\mathcal{B}(u)$ and $\mathcal{B}(w)$ into $\mathcal{B}$, copying the boundary points of $\mathcal{B}(v)$ to the lists, and then removing $\mathcal{B}(v)$ from $\mathcal{B}$. By Corollary 5, h_v intersects $O(m)$ cells in M_{y^+}. Thus, $\mathcal{B}(v)$ has $O(m)$ boundary points, and the update to $\mathcal{B}(u)$ and $\mathcal{B}(w)$ takes $O(m)$ time. There is no change to distance functions.

Since there are n bottom-side events, it takes $O(nm + n \log n)$ time to update $\mathcal{T}$ and $O(nm)$ time to update $\mathcal{B}$ for all bottom-side events.

▶ **Lemma 7.** *We can handle all bottom-side events in $O(nm + n \log n)$ time using $O(nm)$ space.*

4.3 Handling a top-side event

When L encounters the top side of a rectangle $R \in \mathsf{R}$, the two consecutive segments in $\mathsf{F} \cap L$ incident to R are replaced by one segment spanning them by the event. See Figure 4(c). We update $\mathcal{T}$ by finding the two nodes $u, w \in \mathcal{T}$ with $x_2(u) = x_1(R)$ and $x_1(w) = x_2(R)$, removing u and w from $\mathcal{T}$, and then inserting a new node v into $\mathcal{T}$. We set $x_1(v) = x_1(u)$, $x_2(v) = x_2(w)$, and $X_v[i] = X_u[i] \vee X_w[i]$ for each $i = 1, \ldots, m$. This takes $O(m + \log n)$ time.

We update the distance function $d_v(x)$ for x with $x_1(v) \leq x \leq x_1(R)$ as follows. The geodesic path from any point $p(x) \in h_u$ to s_i with $X_u[i] = \mathtt{F}$ and $X_w[i] = \mathtt{T}$ is xy-monotone by Lemma 1, and thus $d(s_i, p(x)) = y(p(x)) - y(s_i) + |x(s_i) - x|$. Also, we observe that $x(s_i) \geq x$ for any x. Thus, every $p(x)$ has the same site s^* as its farthest site among the sites s_i with $X_u[i] = \mathtt{F}$ and $X_w[i] = \mathtt{T}$. Then $d(s^*, p(x)) = y(p(x)) - y(s^*) + x(s^*) - x$. By Lemma 4, there is at most one maximal interval I of $x \in [x_1(v), x_1(R)]$ such that $d_v(x) \leq d(s^*, p(x))$. Moreover, I is bounded from left by $x_1(v)$. We find the boundary point $p(x^*) \in h_u$ such that $d_v(x^*) = d(s^*, p(x^*))$, and update $d_v(x)$ to $d(s^*, p(x))$ for $x \leq x^*$.

If there is no such point $p(x^*)$, either $d_v(x) < d(s^*, p(x))$ or $d_v(x) > d(s^*, p(x))$ for all x with $x_1(v) \leq x \leq x_1(R)$. If $d_v(x) < d(s^*, p(x))$, we update $d_v(x)$ to $d_v(x) = d(s^*, p(x))$ for $x_1(v) \leq x \leq x_1(R)$. If $d_v(x) > d(s^*, p(x))$, we do not update $d_v(x)$.

We update $\mathcal{B}[x_1(v), x_1(R)]$, which is a part of $\mathcal{B}(v)$ with range $[x_1(v), x_1(R)]$, by removing all the boundary points in the interior of I in time linear to the number of the boundary points, and then inserting $p(x^*)$ as a boundary point. We can handle the case of x with $x_2(R) \leq x \leq x_2(v)$, and update $\mathcal{B}[x_2(R), x_2(v)]$ analogously.

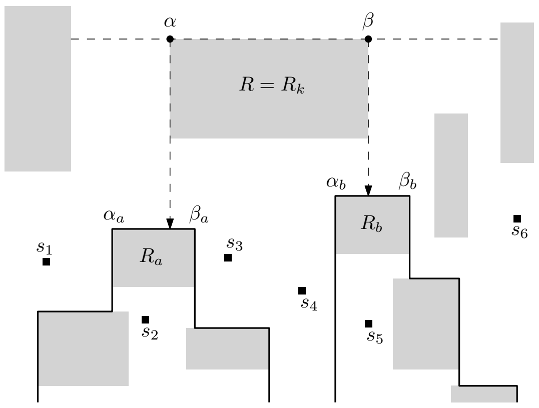

Figure 5 $\mathsf{S}^T = \{s_1, s_2, s_3, s_4, s_5, s_6\}$ is partitioned into $\mathsf{S}_k = \{s_2, s_3, s_4, s_5\}$, $\mathsf{S}(\alpha) = \{s_1\}$, and $\mathsf{S}(\beta) = \{s_6\}$. For two rectangles R_a and R_b, $\mathsf{S}_a = \{s_2\}$ and $\mathsf{S}_b = \{s_5\}$.

Computing distance functions for a top side

We show how to compute $d_v(x)$ for $x \in [x_1(R), x_2(R)]$ and update $\mathcal{B}[x_1(R), x_2(R)]$ efficiently.

For an index k, let α_k and β_k denote the top-left corner and the top-right corner of $R_k \in \mathsf{R}$, and let S_k denote the set of the sites that lie below the polygonal curve consisting of $\pi_{\mathsf{dl}}(\alpha_k)$, the top side of R_k, and $\pi_{\mathsf{dr}}(\beta_k)$.

For the top-side event of $R = R_k$, let $\alpha = \alpha_k$ and $\beta = \beta_k$. Note that $x(\alpha) = x_1(R)$ and $x(\beta) = x_2(R)$. Let S^T be the set of the sites s_i, with $X_v[i] = \mathtt{T}$ for all $i = 1, \ldots, m$. We partition S^T into three disjoint subsets, S_k, $\mathsf{S}(\alpha)$, and $\mathsf{S}(\beta)$, such that $\mathsf{S}(\alpha) = \{s_i \in \mathsf{S}^T \setminus \mathsf{S}_k \mid x(s_i) \leq x_1(R)\}$ and $\mathsf{S}(\beta) = \{s_i \in \mathsf{S}^T \setminus \mathsf{S}_k \mid x(s_i) \geq x_2(R)\}$. See Figure 5.

Every geodesic path from any site in $\mathsf{S}(\alpha)$ or $\mathsf{S}(\beta)$ to any point on the top side of R is xy-monotone. Thus for any point $p(x)$ lying on the top side of R, we can compute $d(s^\alpha, p(x))$ and $d(s^\beta, p(x))$, where s^α and s^β are the farthest sites of $p(x)$ among sites in $\mathsf{S}(\alpha)$ and among sites in $\mathsf{S}(\beta)$, respectively, as we did for $\mathcal{B}[x_1(v), x_1(R)]$ or $\mathcal{B}[x_2(R), x_2(v)]$.

We denote by $d_\alpha(i, x) = d(\alpha, s_i) + x - x(\alpha)$ the length of a geodesic path from a site s_i to $p(x)$ passing through α, and denote by $d_\beta(i, x) = d(\beta, s_i) + x(\beta) - x$ the length of a geodesic path from s_i to $p(x)$ passing through β. Let $D(x) = \max_{s_i \in \mathsf{S}_k} \min\{d_\alpha(i, x), d_\beta(i, x)\}$ for all x with $x(\alpha) \leq x \leq x(\beta)$. Then $d_v(x) = \max\{d(s^\alpha, p(x)), D(x), d(s^\beta, p(x))\}$. Thus, once we compute $D(x)$ in $O(m)$ time, we can compute $d_v(x)$ in time linear to the complexity of $D(x)$, which is $O(m)$. To compute $D(x)$, we find the two rectangles hit first by the vertical rays, one emanating from α and one emanating from β, going downwards. Using these two rectangles we compute the distance functions $d(\alpha, s_i)$ and $d(\beta, s_i)$ for all $s_i \in \mathsf{S}_k$. Using these distance functions, we can compute $D(x)$ in $O(m)$ time. Details are given in the full version. We update $\mathcal{B}[x_1(R), x_2(R)]$ in $O(m)$ time using $d_v(x)$.

There are n top-side events, so we can handle the top-side events in $O(nm + n \log n)$ time. In addition, we compute distances from $O(m)$ sites to each corner of $O(n)$ rectangles, and store them. Using ray shooting queries emanating from the corners of rectangles, it takes $O(nm) + O(n \log n)$ time using $O(nm)$ space. Therefore, we have the following lemma.

▶ **Lemma 8.** *We can handle all top-side events in $O(nm + n \log n)$ time using $O(nm)$ space.*

4.4 Constructing the query data structure

Initially, $\mathsf{Q}_{y^+} = \emptyset$. For each site event and top-side event, we update $d_v(x)$ and $\mathcal{B}(v)$ for node v of $\mathcal{T}$ corresponding to the event. We insert a horizontal segment h corresponding to each interval which is updated at the event into Q_{y^+}, and copy the boundary points into h. For each site event, at most one horizontal line segment h is inserted. There is no boundary point in the interior of h, so we can copy h with two endpoints in $O(1)$ time. For each top-side event, at most three horizontal line segments are inserted. They have $O(m)$ boundary points by Lemma 4, so we can copy them in $O(m)$ time. There are $O(n+m)$ horizontal segments and $O(nm)$ boundary points in Q_{y^+}, so the query structure Q_{y^+} uses $O(nm)$ space.

Farthest-point queries

Once Q_{y^+} is constructed, we can find $f_{y^+}(q)$ from a query point $q \in \mathsf{F} \setminus C_\emptyset$. We find the farthest sites from q in the other three maps using their query data structures.

By Corollary 3, our query problem reduces to the vertical ray shooting queries. We use the data structure by Giora and Kaplan [15] for vertical ray shooting queries on $O(n+m)$ horizontal line segments in Q_{y^+}, which requires $O((n+m)\log(n+m))$ time and $O(n+m)$ space for construction. Let h be the horizontal segment in Q_{y^+} hit first by the vertical ray emanating from q going downwards. We can find h in $O(\log(n+m))$ time using the ray shooting structure. If no horizontal segment in Q_{y^+} is hit by the ray, q is y^+-reachable from no site. Otherwise, there are $O(m)$ boundary points on h, sorted in increasing order of x-coordinate. With those boundary points, we can find $f(q)$ for a query point q in $O(\log m)$ time using binary search. Thus, a farthest-neighbor query takes $O(\log(n+m))$ time in total.

Once the farthest sites of q for each of the four data structures is found, we take the sites with the largest distance among them as the farthest sites $f(q)$ of S from q. Combining Lemmas 6, 7 and 8 with query time, we have the following theorem.

▶ **Theorem 9.** *We can construct a data structure for m point sites in the presence of n axis-aligned rectangular obstacles in the plane in $O(nm + n\log n + m\log m)$ time and $O(nm)$ space that answers any L_1 farthest-neighbor query in $O(\log(n+m))$ time.*

5 Computing the Explicit Farthest-point Voronoi Diagram

We construct the explicit farthest-point Voronoi diagram $\mathsf{FVD} = \mathsf{FVD}(\mathsf{S}, \mathsf{R})$ of a set S of m point sites in the presence of a set R of n rectangular obstacles in the plane. It is known that FVD requires $\Omega(nm)$ space [5, 7]. It takes $\Omega(n\log n)$ time to compute the geodesic distance between two points in F [12]. By a reduction from the sorting problem, it can be shown to take $\Omega(m\log m)$ time for computing the farthest-point Voronoi diagram of m point sites in the plane. We present an $O(nm + n\log n + m\log m)$-time algorithm using $O(nm)$ space that matches the time and space lower bounds. This is the first optimal algorithm for constructing the farthest-point Voronoi diagram of points in the presence of obstacles in the plane in both time and space.

We construct Q_{y^+} using the plane sweep in Section 4. During the plane sweep, we find all horizontal edges of M_{y^+} and insert them into Q_{y^+} as segments. We find all the lower endpoints of the vertical edges of M_{y^+} and insert them as boundary points in $\mathcal{B}$. We also find the upper endpoints of vertical edges of M_{y^+}. By connecting those endpoints using vertical segments appropriately, we can construct M_{y^+} from Q_{y^+} in a doubly connected edge list without increasing the time and space complexities. The other three maps can also be constructed in the same way in the same time and space.

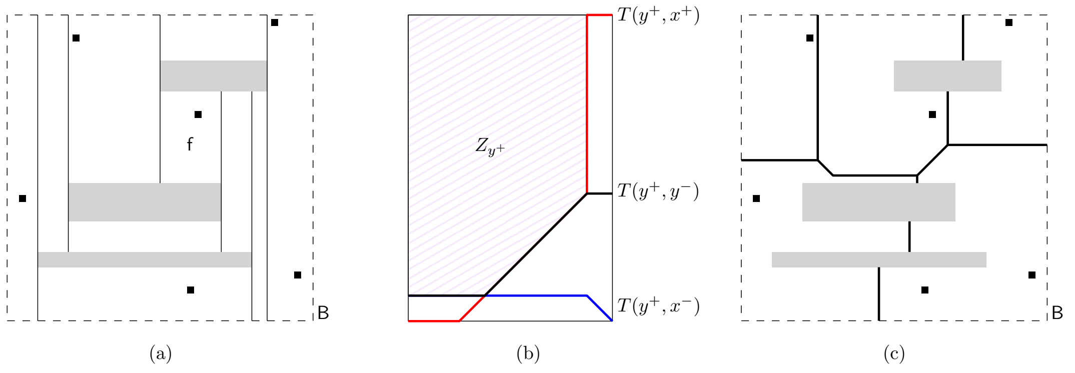

Figure 6 (a) Vertical decomposition F_V. f is a face of F_V. (b) Z_{y^+} is a region in f above the upper envelope of three traces, $T(y^+, y^-)$, $T(y^+, x^+)$ and $T(y^+, x^-)$. (c) Explicit geodesic L_1 farthest-point Voronoi diagram FVD.

We construct the farthest-point Voronoi diagram FVD using the four maps explicitly. Note that $f(p) = f_{y^+}(p)$ for any point p lying on the top side of B. Thus, it suffices to compute FVD in $\mathsf{F} \cap \mathsf{B}$. For ease of description, we assume that the x-coordinates of the rectangles in R are all distinct. We consider a vertical decomposition F_V obtained by drawing maximal vertical line segments contained in $\mathsf{F} \cap \mathsf{B}$ of which each is extended from a vertical side of a hole of F. Let V be a set of such vertical line segments. $\mathsf{F} \setminus \bigcup_{\ell \in V} \ell$ consists of $O(n)$ connected faces. Each face is a rectangle since each hole of F is a rectangle and F is bounded by B. See Figure 6(a).

Any two farthest-point maps $\mathsf{M}_1, \mathsf{M}_2$ have a *bisector* which consists of the points in F having the same distance to their farthest sites in M_1 and in M_2. The four maps define six bisectors. In a face of F_V, the six bisectors and some axis-aligned segments partition the face into *zones* such that FVD restricted to one zone coincides with the diagram in the corresponding region of a farthest-point map. Thus, we compute the bisectors between maps in each face of F_V, partition the face into zones, find the region of a farthest-point map corresponding to each zone, and then glue the regions and faces to compute FVD completely.

5.1 Bisectors of farthest-point maps

We define the *bisector* between M_δ and $\mathsf{M}_{\delta'}$ as $B(\delta, \delta') = \{q \in \mathsf{F} \mid d_\delta(q) = d_{\delta'}(q)\}$ for any two distinct $\delta, \delta' \in \{y^+, y^-, x^+, x^-\}$. We show that any vertical line intersects $B(y^+, y^-)$ in at most one point, and any vertical line segment contained in F intersects $B(y^+, x^+)$ (and $B(y^+, x^-)$) in at most one connected component. Thus, these three bisectors contained in a face of F_V are x-monotone. Details are given in the full version.

For each face f of F_V, we compute the portion of $B(y^+, y^-)$ contained in f. As $B(y^+, y^-) \cap \mathsf{f}$ is x-monotone, we sweep a vertical line L from $x_1(\mathsf{f})$ to $x_2(\mathsf{f})$ maintaining a point $p \in \mathsf{f} \cap L$ with $d_{y^+}(p) = d_{y^-}(p)$. First, we compute p lying on the left side of f as follows. There are $O(m)$ intersections of the left side of f with the horizontal segments of Q_{y^+} and Q_{y^-} as any vertical line segment contained in F intersects $O(m)$ horizontal segments of them. For each intersection point q, we compute $d_{y^+}(q)$ and $d_{y^-}(q)$, and find two consecutive points q_1 and q_2 among the intersection points by y-coordinate such that $d_{y^+}(q_1) \leq d_{y^-}(q_1)$ and $d_{y^-}(q_2) \leq d_{y^+}(q_2)$. We can compute q_1 and q_2 in $O(m)$ time using Q_{y^+} and Q_{y^-}. Then we compute p lying on q_1q_2.

Having the distance functions, we have the slope of the bisector incident to p. Let $\vec{\ell}$ be the half-line from p with the slope going rightward. We find the first point p' on $\vec{\ell}$ from p at which the slope of $d_{y^+}(p')$ or $d_{y^-}(p')$ changes. Since the slope of $d_{y^+}(p')$ changes at most once within a cell of M_{y^+}, we can find p' in time linear to the complexity of the cells containing p of the maps. If there are two or more such points, p is the point with the maximum y-coordinate among them.

There may be no point p satisfying $d_{y^+}(p) = d_{y^-}(p)$ if there is a point $q \in \mathsf{f} \cap L$ such that $d_{y^+}(q') > d_{y^-}(q')$ for every point q' lying above q, and $d_{y^+}(q') < d_{y^-}(q')$ for every point q' lying below q. We maintain the point q in this case. Note that q follows a horizontal segment during the plane sweep, and thus we can find the first point p with $d_{y^+}(p) = d_{y^-}(p)$ using a horizontal half-line from q.

During the plane sweep, p or q moves along $B(y^+, y^-)$ rightwards until it meets the right side of f. We compute the other bisectors in f similarly.

We compute the *trace* $T(y^+, y^-)$ of p and q during the sweep. Observe that every vertical line intersecting f also intersects the trace in one point t. Moreover, if the line intersects $B(y^+, y^-) \cap \mathsf{f}$, t is the topmost point of the intersection. Since we have M_{x^+} and M_{x^-}, we can compute the two traces $T(y^+, x^+)$ and $T(y^+, x^-)$ similarly.

We observe that each bisector and trace in f has $O(m)$ complexity. We get the distance functions using Q_{y^+}, Q_{y^-}, Q_{x^+}, and Q_{x^-} which consist of $O(n+m)$ line segments and support $O(\log(n+m))$ query time. After computing those distance functions, the traces can be constructed in time linear to their complexities. Thus, in total it takes $O(nm + n\log n + m\log m)$ time to construct the traces for all faces.

5.2 Partitioning f into zones

With the three traces $T(y^+, y^-)$, $T(y^+, x^+)$, $T(y^+, x^-)$ in f, we compute the zone Z_{y^+} in f corresponding to M_{y^+} in f. Let T be an upper envelope of $T(y^+, y^-)$, $T(y^+, x^+)$ and $T(y^+, x^-)$. Then Z_{y^+} is the set of points lying above T in f. See Figure 6(b). The following lemma can be shown using the lemmas in the full version.

▶ **Lemma 10.** *For any point $p \in Z_{y^+}$, $f(p) = f_{y^+}(p)$.*

Similarly, we define the other three zones Z_{y^-}, Z_{x^+}, and Z_{x^-}. Note that $d_\delta(p) > d_{\delta'}(p)$ for every point $p \in Z_\delta$ for distinct $\delta, \delta' \in \{y^+, y^-, x^+, x^-\}$. By Lemma 10, $\mathsf{FVD} \cap Z_{y^+}$ coincides with M_{y^+}. We copy the corresponding farthest-point map of δ into Z_δ for each $\delta \in \{y^+, y^-, x^+, x^-\}$.

We call $\mathsf{f} \setminus (Z_{y^+} \cup Z_{y^-} \cup Z_{x^+} \cup Z_{x^-})$ the bisector zone. Every point p in the bisector zone lies on a bisector of two or more maps. Thus, for each bisector of two maps, we copy one of the maps into the corresponding zone.

5.3 Gluing along boundaries

We first glue the zones along their boundaries in each face of F_V. For each edge e incident to two zones, we check whether the two cells incident to the edge have the same farthest site or not. If they have the same farthest site, e is not a Voronoi edge of FVD. Then we remove the edge and merge the cells into one. If they have different farthest sites, e is a Voronoi edge of FVD. This takes $O(nm)$ time in total, which is linear to the number of Voronoi edges and cells in FVD.

After gluing zones in every face, we glue the faces of F_V along their boundaries. Since e is a vertical line segment and incident to more than two cells, we divide e into pieces such that any point in the same piece e' is incident to the same set of two cells. If both cells incident to e' have the same farthest site, e' is not a Voronoi edge of FVD. Then we remove the edge and merge the cells. If they have different farthest sites, e' is a Voronoi edge of FVD. There are $O(n)$ vertical line segments in V and each of them intersects $O(m)$ cells of FVD, so it takes $O(nm)$ time in total. Then we obtain the geodesic L_1 farthest-point Voronoi diagram FVD explicitly. See Figure 6(c).

▶ **Theorem 11.** *We can compute the L_1 farthest-point Voronoi diagram of m point sites in the presence of n axis-aligned rectangular obstacles in the plane in $O(nm + n \log n + m \log m)$ time and $O(nm)$ space.*

▶ **Corollary 12.** *We can compute the L_1 geodesic center of m point sites in the presence of n axis-aligned rectangular obstacles in the plane in $O(nm + n \log n + m \log m)$ time and $O(nm)$ space.*

6 Concluding Remarks

We present an optimal algorithm for computing the farthest-point Voronoi diagram of point sites in the presence of rectangular obstacles. However, our algorithm may not work for more general obstacles as it is, because some properties we use for the axis-aligned rectangles including their convexity may not hold any longer. Our results, however, may serve as a stepping stone to closing the gap to the optimal bounds.

References

1 A. Aggarwal, L.J. Guibas, J. Saxe, and P.W. Shor. A linear-time algorithm for computing the Voronoi diagram of a convex polygon. *Discrete & Computational Geometry*, 4(6):591–604, 1989.

2 H. Alt, O. Cheong, and A. Vigneron. The Voronoi diagram of curved objects. *Discrete & Computational Geometry*, 34(3):439–453, 2005.

3 B. Aronov. On the geodesic Voronoi diagram of point sites in a simple polygon. *Algorithmica*, 4(1):109–140, 1989.

4 B. Aronov, S. Fortune, and G. Wilfong. The furthest-site geodesic Voronoi diagram. *Discrete & Computational Geometry*, 9(3):217–255, 1993.

5 S.W. Bae and K.-Y. Chwa. The geodesic farthest-site Voronoi diagram in a polygonal domain with holes. In *Proceedings of the 25th Annual Symposium on Computational Geometry (SoCG)*, pages 198–207, 2009.

6 B. Ben-Moshe, B.K. Bhattacharya, and Q. Shi. Farthest neighbor Voronoi diagram in the presence of rectangular obstacles. In *Proceedings of the 13th Canadian Conference on Computational Geometry (CCCG)*, pages 243–246, 2005.

7 B. Ben-Moshe, M.J. Katz, and J.S.B. Mitchell. Farthest neighbors and center points in the presence of rectangular obstacles. In *Proceedings of the 17th Annual Symposium on Computational Geometry (SoCG)*, pages 164–171, 2001.

8 O. Cheong, H. Everett, M. Glisse, J. Gudmundsson, S. Hornus, S. Lazard, M. Lee, and H.-S. Na. Farthest-polygon Voronoi diagrams. *Computational Geometry*, 44(4):234–247, 2011.

9 L.P. Chew and R.L. Dyrsdale III. Voronoi diagrams based on convex distance functions. In *Proceedings of the 1st annual symposium on Computational geometry (SoCG)*, pages 235–244, 1985.

10 J. Choi, C.-S. Shin, and S.K. Kim. Computing weighted rectilinear median and center set in the presence of obstacles. In *International Symposium on Algorithms and Computation*, pages 30–40. Springer, 1998.

11 J. Choi and C. Yap. Monotonicity of rectilinear geodesics in d-space. In *Proceedings of the 12th Annual Symposium on Computational Geometry (SoCG)*, pages 339–348, 1996.

12 P.J. De Rezende, D.-T. Lee, and Y.-F. Wu. Rectilinear shortest paths with rectangular barriers. In *Proceedings of the 1st Annual Symposium on Computational Geometry (SoCG)*, pages 204–213, 1985.

13 H. Edelsbrunner and R. Seidel. Voronoi diagrams and arrangements. *Discrete & Computational Geometry*, 1(1):25–44, 1986.

14 S. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2(1):153–174, 1987.

15 Y. Giora and H. Kaplan. Optimal dynamic vertical ray shooting in rectilinear planar subdivisions. *ACM Transactions on Algorithms*, 5(3):28:1–51, 2009.

16 J. Hershberger and S. Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM Journal on Computing*, 28(6):2215–2256, 1999.

17 R. Klein. Abstract Voronoi diagrams and their applications. In *Proceedings of the 4th International Workshop on Computational Geometry (EuroCG)*, pages 148–157. Springer, 1988.

18 D.-T. Lee. Two-dimensional Voronoi diagrams in the L_p-metric. *Journal of the ACM*, 27(4):604–618, 1980.

19 E. Oh. Optimal algorithm for geodesic nearest-point Voronoi diagrams in simple polygons. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 391–409, 2019.

20 E. Oh and H.-K. Ahn. Voronoi diagrams for a moderate-sized point-set in a simple polygon. *Discrete & Computational Geometry*, 63(2):418–454, 2020.

21 E. Oh, L. Barba, and H.-K. Ahn. The geodesic farthest-point Voronoi diagram in a simple polygon. *Algorithmica*, 82(5):1434–1473, 2020.

22 E. Papadopoulou and S.K. Dey. On the farthest line-segment Voronoi diagram. *International Journal of Computational Geometry & Applications*, 23(06):443–459, 2013.

23 E. Papadopoulou and D.T. Lee. The L_∞ Voronoi diagram of segments and VLSI applications. *International Journal of Computational Geometry & Applications*, 11(05):503–528, 2001.

24 M.I. Shamos and D. Hoey. Closest-point problems. In *Proceedings of the 16th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 151–162, 1975.

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