Efficient Computation of Rectilinear Geodesic Voronoi Neighbor in the Presence of Obstacles

Pinaki Mitra

Department of Computer Science and Engineering, Jadavpur University, Calcutta 700 032, India

and

Subhas C. Nandy*

Indian Statistical Institute, Calcutta 700 035, India

Received February 5, 1997; revised March 29, 1998

In this paper we present an algorithm to compute the rectilinear geodesic voronoi neighbor of an arbitrary query point q among a set S of m points in the presence of a set $\mathscr{O}$ of n vertical line segment obstacles inside a rectangular floor. The distance between a pair of points α and β is the shortest rectilinear distance avoiding the obstacles in $\mathscr{O}$ and is denoted by $\delta(\alpha, \beta)$. The rectilinear geodesic voronoi neighbor of an arbitrary query point q, $RGVN(q)$ is the point $p_i \in S$ such that $\delta(q, p_i)$ is minimum. The algorithm suggests a preprocessing of the elements of the sets S and $\mathscr{O}$ in $O((m+n)\log(m+n))$ time such that for an arbitrary query point q, the $RGVN$ query can be answered in $O(\log(m+n))$ time. The space required for storing the preprocessed information is $O(n + m \log m)$. If the points in S are placed on the boundary of the rectangular floor, a different technique is adopted to decrease the space complexity to $O(m+n)$. This technique works even if the obstacles are rectangles instead of line segments. Finally, the parallelization of the preprocessing steps for the latter algorithm is suggested, which takes $O(\log^3(m+n))$ time, using $O((m+n)^{1.5}/\log^2(m+n))$ processors and $O(\log(m+n))$ query time.

LIST OF SYMBOLS

Set of points, called voronoi sites.
Cardinality of the set S.
Set of obstacles.

n	Cardinality of the set $\mathcal{O}$.
q	Arbitrary query point.
p, α, and β	Refer to arbitrary points on the plane.
p_x	The x coordinate of point p.
p_y	The y coordinate of point p.
$\pi(\alpha, \beta)$	Shortest rectilinear path between the pair of points α and β avoiding the obstacles in $\mathcal{O}$.
$\delta(\alpha, \beta)$	The length of $\pi(\alpha, \beta)$.
$L_1(\alpha, \beta)$	The length of the rectilinear shortest path among the points α and β in the absence of the obstacles. Note that, in this situation, the rectilinear shortest path is monotone in both the x and y directions, and is an L-shaped path.
$\mathcal{P}_q$	The set of points in S such that the shortest rectilinear distance from q to any two members in $\mathcal{P}_q$ avoiding the obstacles in $\mathcal{O}$ are same, and is less than the shortest rectilinear distance of any other point in $S - \mathcal{P}_q$ from q.
$RGVN(q)$	The rectilinear geodesic voronoi neighbor of the query point q.
X^+Y^+ chain	A chain consisting of alternating horizontal towards right and upward vertical line segments from point p to the top right corner of the floor avoiding the obstacles, where the first move from point p is in horizontal direction, and the direction changes when the move hits the boundary of an obstacle or the boundary of the floor. The X^+Y^-, X^-Y^+, X^-Y^-, Y^+X^+, Y^+X^-, Y^-X^+, Y^-X^- chains can also be defined in a similar way.
R_{x^+}	Free space enclosed by X^+Y^+, X^+Y^- chains (it is a staircase region). The R_{x^-}, R_{y^+}, and R_{y^-} regions can also be defined in a similar way.
$R_{x^+y^+}$	Free space enclosed by X^+Y^+, Y^+X^+ chains. The $R_{x^-y^-}$ region is defined in a similar way.
$R_{x^+y^-}$	Free space enclosed by X^+Y^-, Y^-X^+ chains. The $R_{x^-y^+}$ region is defined in a similar way.
$\mathcal{L}$	The sweep line.
C	The set of end points of the obstacles in $\mathcal{O}$.
$LNN(u_i)$	The left nearest neighbor $p_j \in S$ of the point $u_i \in C$.
$RNN(u_i)$	The right nearest neighbor $p_j \in S$ of the point $u_i \in C$.
$RGVN(u_i)$	It is $LNN(u_i)$ if $\delta(u_i, LNN(u_i)) \leq \delta(u_i, RNN(u_i))$; otherwise it is $RNN(u_i)$.
C'	The set of end points in C already encountered by $\mathcal{L}$ at a particular instant.

S'	The set of points in S already encountered by $\mathscr{L}$ at a particular instant.
$\mathscr{C}$	$C \cup S$.
$\mathscr{C}'$	$C' \cup S'$.
I_i	The interval on the sweep line $\mathscr{L}$ such that any point on I_i is closer to p_i than other points.
$Q[x_I, x_J][y_K, y_L]$	The query rectangle whose horizontal span is $[x_I, x_J]$ and the vertical span is $[y_K, y_L]$, where either $y_K = y_{\min}$ or $y_L = y_{\max}$.
S^*	List of points inside a query rectangle.
$+X$ direction	To refer to the right side of a point q, we introduced the term $+X$ direction of point q.

1. INTRODUCTION

Given a set S of m points, called *voronoi sites*, placed inside a bounding rectangular region that contains a set $\mathscr{O}$ of n obstacles, the rectilinear geodesic voronoi/nearest neighbor $RGVN(q)$ of an arbitrary query point q is defined to be the set of points $\mathscr{P}_q \subseteq S$ such that the shortest rectilinear distance from q to any member $p_i \in \mathscr{P}_q$ avoiding the obstacles in $\mathscr{O}$, denoted by $\delta(q, p_i)$, is less than the shortest rectilinear distance of any other point $p_j \in S \setminus \mathscr{P}_q$ from q. In addition, for any two points p_i and $p_j \in \mathscr{P}_q$, $\delta(q, p_i) = \delta(q, p_j)$ (provided the number of points in $\mathscr{P}_q$ is more than 1). Given a query point, our goal is to locate any point from $\mathscr{P}_q$. This will be referred to as an $RGVN$-query. In [7], an algorithm is presented in which the obstacles are isothetic nonoverlapping rectangles. It computes the rectilinear geodesic voronoi diagram in $O((m+n)\log(m+n)\log n)$ time. A more general problem of computing a rectilinear geodesic voronoi diagram, in which the obstacles are arbitrary polygons with a total of $O(n)$ vertices, is studied in [14]. The preprocessing time for their algorithm is $O((m+n)\log^2(m+n))$. After construction of the rectilinear geodesic voronoi diagram, both of the algorithms can report $RGVN(q)$ for any arbitrary query point q in $O(\log(m+n))$ time. Such types of geometric problems often arise in VLSI chip design, robot motion planning, mobile computing, to name a few. In this paper, we have shown that the preprocessing time can be reduced in the following two restricted cases of the problem:

P1 The obstacles are vertical line segments, and voronoi sites are placed arbitrarily on the rectangular floor.

P2 The obstacles are nonoverlapping isothetic rectangles, and the voronoi sites are located on the boundary of the rectangular floor.

In both problems, instead of computing the rectilinear geodesic voronoi diagram, we build up an appropriate data structure in the preprocessing phase to answer efficiently the $RGVN$ queries for any arbitrary query point.

The preprocessing phase for the first problem requires $O((m+n)\cdot \log(m+n))$ time. It uses line sweep paradigm and maintains a layered segment tree that requires $O(n + m \log m)$ space. It allows us to solve the $RGVN$ query for an arbitrary query point q among the members in S, in $O(\log(m+n))$ time.

The method of preprocessing for the second problem is based on Mehlhorn's [13] algorithm for computing the voronoi neighbors of a subset of vertices on the *carrier graph* [6] for the set of rectangles in $\mathscr{E}$. This step requires $O(m+n)$ space and $O((m+n)\log(m+n))$ time. Here also the $RGVN$ query for an arbitrary query point can be answered in $O(\log(m+n))$ time.

The paper is organized as follows. In Section 2, some practical applications of both problems are rigorously discussed. Section 3 introduces some terminology and preliminary results. Sections 4 and 5 deal with problems P1 and P2, respectively. Finally, a discussion about some open problems and the concluding remarks appear in Section 6.

2. APPLICATIONS

Rectilinear geodesic voronoi neighbors for query points among a set of disjoint isothetic rectangular barriers may have many applications in VLSI chip design.

Consider a VLSI routing problem in which a rectangular chip floor contains a set of disjoint rectangular circuit modules. A set of nets is attached to each circuit module. Terminals of a given net may appear in several modules that are to be electrically interconnected. Let a new module now be placed on the chip floor, and each net associated with it is to be connected to the nearest pin of the same net, already present on the chip floor. This leads to the problem posed by Guha and Suzuki [7]. Here the circuit modules may be considered as rectangular obstacles; a given net attached to the newly inserted block is the query point. Connection is to be implemented using a rectilinear path to the nearest pin of the same net; thus the pins of that net already placed on the chip floor may be considered as voronoi sites.

In particular, if a newly arrived pin needs to be connected to any of the existing pins of the same net appearing on the boundary of the chip floor, it means that the voronoi sites appear along the boundary of the chip floor

and the rectangular circuit modules play the role of obstacles. Thus the job leads to the solution of problem P2 posed in this paper.

Problem P1 may evolve from routing problems in multichip modules (MCMs). Inside an MCM, the circuit components are mounted on the top layer, and the other layers are used as routing layers. The pins for electrical connections, attached to a circuit component, are accessible from all layers along a pair of vertical lines. Thus the pins exposed on the routing layers appear along a line and are closed enough so that no routing wire can pass through them. These vertical lines may be considered as obstacles. A pin corresponding to a particular net needs to be connected to some other pin of the same net by using isothetic wire segments. The length of this routing wire needs to be as small as possible to reduce the routing congestion and the time delay for signal propagation. This can be formulated by problem P1.

Both of the problems discussed in this paper may also find several applications in the area of plant and facility layout, urban transportation, locating power lines, robot motion planning, mobile communication, to name a few [10].

3. PRELIMINARIES

In this section, we review some existing results on the rectilinear shortest path problem in the presence of a set of isothetic nonoverlapping rectangular obstacles. Throughout our discussion we assume that the floor on which the obstacles and the voronoi sites are distributed is rectangular. The query point is assumed to be inside the floor. We shall denote the x and y coordinates of a point p by p_x and p_y, respectively. For a pair of points p and q on the floor, the rectilinear shortest path among them that avoids the obstacles is denoted by $\pi(p, q)$, and its length is denoted by $\delta(p, q)$. Surely if the obstacles are not present, this path becomes L-shaped, consisting of a single horizontal and a single vertical line segment, and its length becomes $L_1(p, q) = |p_x - q_x| + |p_y - q_y|$.

DEFINITION 1. Given the set $\mathcal{O}$ of n rectangular obstacles, an X^+Y^+ *chain* from a point p is a path from p to the top right corner of the floor obtained as follows: Start from the point p and move horizontally to the right side until a rectangle in $\mathcal{O}$ or the boundary of the floor is hit; then move vertically up until the obstacle is cleared or the top right corner of the floor is reached. In the former case, i.e., if an obstacle is encountered, horizontal motion is resumed. This process is repeated until the top right corner of the floor is reached.

In a similar way, we can define X^+Y^-, X^-Y^+, X^-Y^-, Y^+X^+, Y^+X^-, Y^-X^+, Y^-X^- chains (see Fig. 1 for an illustration). It must be mentioned that the difference between the X^+Y^+ chain and the Y^+X^+ chain is that the X^+Y^+ (resp. Y^+X^+) chain originates from point p with a horizontal (resp. vertical) line segment.

Note that when the obstacles are vertical line segments, the Y^+X^+ (resp. Y^+X^-) chain is a vertical line segment from p to the top boundary of the floor, followed by a horizontal line segment up to the top right (resp. top left) corner of the floor along its top boundary. Similarly, the Y^-X^+ (resp. Y^-X^-) chain is a vertical line segment from p to the bottom boundary of the floor, followed by a horizontal line segment up to the bottom right (resp. bottom left) corner of the floor along its bottom boundary.

DEFINITION 2. The free space on the floor may be partitioned into following eight regions, as illustrated in Fig. 1.

R_{x^+} region: Free space enclosed by X^+Y^+ and X^+Y^- chains.
R_{x^-} region: Free space enclosed by X^-Y^+ and X^-Y^- chains.
R_{y^+} region: Free space enclosed by Y^+X^+ and Y^+X^- chains.
R_{y^-} region: Free space enclosed by Y^-X^+ and Y^-X^- chains.
$R_{x^+y^+}$ region: Free space enclosed by X^+Y^+ and Y^+X^+ chains.
$R_{x^-y^+}$ region: Free space enclosed by Y^+X^- and X^-Y^+ chains.
$R_{x^-y^-}$ region: Free space enclosed by X^-Y^- and Y^-X^- chains.
$R_{x^+y^-}$ region: Free space enclosed by X^+Y^- and Y^-X^+ chains.

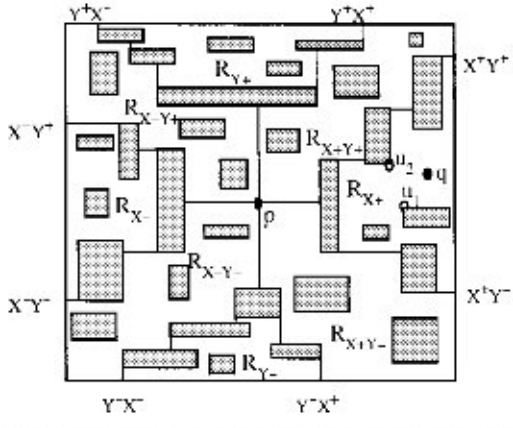


FIG. 1. Demonstration of the properties of rectilinear geodesic shortest path among rectangular obstacles.

It is easy to observe that the R_{x^+}, R_{x^-}, R_{y^+}, and R_{y^-} regions are aligned to the right, left, top, and bottom boundaries of the floor, respectively, and the $R_{x^+y^+}$, $R_{x^-y^+}$, $R_{x^-y^-}$, and $R_{x^+y^-}$ regions span from the point p to, respectively, the top left, top right, bottom right, and bottom left corner points of the floor.

LEMMA 1 [5]. *The shortest path from p to an arbitrary point in the R_{x^+} region is monotonic in the positive direction of the x axis.*

COROLLARY 1 [11]. *The rectilinear shortest path between any pair of points in the presence of vertical line segments is always monotone in the direction of the x axis.*

LEMMA 2 [5]. *There exists a shortest path from p to an arbitrary point q in the R_{x^+} region that passes through one of the two points u_1 and u_2, where u_1 (resp. u_2) is a corner point of some rectangle visible from q, having $u_{1x} < q_x$ (resp. $u_{2x} < q_x$) and $u_{1y} < q_y$ (resp. $u_{2y} > q_y$) such that u_{1y} (resp. u_{2y}) is greater than (resp. less than) the y coordinate of any other corner point r of some rectangle that is visible from q, with $r_x < q_x$, and $r_y < q_y$ (resp. $r_y > q_y$)* (*see Fig.* 1).

Similar results hold for the other seven regions, referred to in Definition 2. It is needless to mention that if the obstacles are vertical line segments, the R_{y^+} and R_{y^-} regions will vanish.

4. RGVN QUERIES IN THE PRESENCE OF LINE SEGMENTS

In problem P1, the set $\mathscr{O}$ of m vertical line segment obstacles and the set S of n voronoi sites are given. In the following two subsections we describe the preprocessing and query answering steps of our algorithm.

4.1. *Preprocessing*

The preprocessing phase of our algorithm consists of two parts: (i) computation of the geodesic voronoi neighbors of each end point of the obstacles in $\mathscr{O}$, and (ii) preprocessing of the set S of voronoi sites, using a layered segment tree data structure.

4.1.1. *Computing Voronoi Neighbors of the End Points of the Obstacles*

In this preprocessing step, a sweep line $\mathscr{L}$ is moved twice, once from the left boundary of the floor to its right boundary, and then from the right boundary of the floor to its left boundary. Let C be the set of end points

of all of the obstacles in $\mathscr{O}$. In the left to right (resp. right to left) sweep, the goal is to compute the left nearest neighbor (LNN) (resp. right nearest neighbor (RNN)) of each end point $u_i \in C$ as defined below:

$$LNN(u_i) = p_j \quad \text{if } u_{i_x} > p_{j_x} \text{ and } \delta(u_i, p_j) \leq \delta(u_i, p_l)\ \forall p_l \in S \text{ such that } u_{i_x} > p_{l_x},$$

$$RNN(u_i) = p_k \quad \text{if } u_{i_x} < p_{k_x} \text{ and } \delta(u_i, p_k) \leq \delta(u_i, p_l)\ \forall p_l \in S \text{ such that } u_{i_x} < p_{l_x}.$$

Finally,

$$\begin{aligned} RGVN(u_i) &= LNN(u_i), && \text{if } \delta(u_i, LNN(u_i)) \leq \delta(u_i, RNN(u_i)) \\ &= RNN(u_i), && \text{otherwise} \end{aligned} \tag{1}$$

In the left-to-right sweep, the points in S and the vertical line segments in $\mathscr{O}$ are processed in sorted order of their x coordinates. The sweep line $\mathscr{L}$ halts when it encounters a member of either S or $\mathscr{O}$. Consider an instant when $S' \subseteq S$ and $O' \subseteq \mathscr{O}$ are, respectively, the voronoi sites and the obstacles already encountered by $\mathscr{L}$. A disjoint set of intervals is maintained on $\mathscr{L}$, where each voronoi interval I_i corresponds to a distinct $p_i \in S'$ such that if we take an arbitrary point $\alpha \in I_i$, then α is closer to $p_i \in S'$ than any other point $p_j \in S'$, $j \neq i$. I_i is called the voronoi interval of p_i on $\mathscr{L}$. The following two lemmas suggest the course of action when (1) $\mathscr{L}$ encounters a voronoi site $p_i \in S - S'$ and (2) $\mathscr{L}$ encounters an obstacle $l_i \in \mathscr{O} - O'$, respectively.

LEMMA 3. *When the sweep line $\mathscr{L}$ encounters a voronoi site p_i, its voronoi interval on $\mathscr{L}$ is single and continuous.*

Proof. [By contradiction]. It is obvious that there is an interval $[a, b]$ around p_i on the sweep line $\mathscr{L}$ such that any point $\alpha \in [a, b]$ will be closer to p_i than any other point $p_j \in S'$. Let β ($\notin [a, b]$) be a point whose geodesic nearest neighbor is p_i, i.e., $\delta(\beta, p_i) < \delta(\beta, p_j)$, $\forall p_j (\neq p_i) \in S'$. We have to show that no such β exists.

Without loss of generality, let us assume that β is below b on $\mathscr{L}$, i.e., $\beta_y < b_y$ (see Fig. 2). Since b is a boundary point of the voronoi interval of p_i, $\delta(b, p_i) = \delta(b, p_j)$ for some $p_j \in S'$. Let us consider a point γ ($b_y > \gamma_y > \beta_y$) in the voronoi interval of p_j. Now $\delta(\beta, p_i) = \overline{p_i\gamma} + \overline{\gamma\beta} > \delta(p_j, \gamma) + \overline{\gamma\beta}$.

Therefore β is nearer to p_j than to p_i. Thus we arrive at a contradiction. ∎

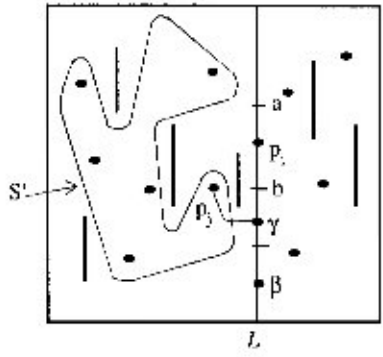


FIG. 2. Proof of Lemma 3.

Thus Lemma 3 suggests that when a voronoi site p_i is encountered by $\mathscr{L}$, the voronoi interval of p_i is to be introduced on $\mathscr{L}$, and the neighboring voronoi intervals are to be modified. The actions to be taken here are mentioned in Step 2.1 of the algorithm Left-to-Right-Sweep and are illustrated in Fig. 5.

LEMMA 4. *Suppose $\mathscr{L}$ is moved to the right from its current position after an encounter with a vertical line $\ell_i[u_1, u_2] \in \mathscr{O}$, $u_1 \in I_i$ and $u_2 \in I_j$. Let u'_1 (resp. u'_2) be the horizontal projection of u_1 (resp. u_2) in the current position of $\mathscr{L}$, i.e., after the encounter with ℓ_i and before encountering any other points of S or obstacles of $\mathscr{O}$, then the left-nearest neighbor of any point $p \in [u'_1, u'_2]$ is either p_i or p_j.*

Proof. [By contradiction]. Let $p \in [u'_1, u'_2]$ has its left-nearest neighbor $p_k \in S'$ (where $p_k \neq p_i \,\&\, p_k \neq p_j$) (see Fig. 3), i.e., $\delta(p, p_k) < \min(\delta(p, p_i), \delta(p, p_j))$. Let P_1 be the shortest path from p_k to p. This path is monotone in the positive direction of the x axis (by Corollary 1). Let θ be a point on the path P_1 having the same x coordinate as that of the line ℓ_i. Without loss of generality, we assume that θ is above u_1.

The length of $P_1 = \delta(p_k, \theta) + \delta(\theta, p) = \delta(p_k, \theta) + L_1(\theta, u_1) + L_1(u_1, p)$.

Now $u_1 \in I_i \Rightarrow \delta(p_k, \theta) + L_1(\theta, u_1) \geq \delta(p_i, u_1)$.

$\Rightarrow$ Length of $P_1 \geq \delta(p_i, u_1) + L_1(u_1, p)$.

This implies that p_i is at least as close to p as p_k is, and it leads to a contradiction. If θ is below u_2, we can similarly show that p_j is closer to p compared to p_k. Thus we arrive at a contradiction. ∎

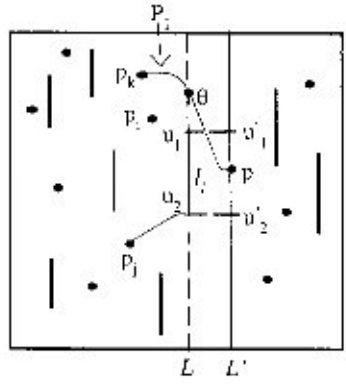


FIG. 3. Proof of Lemma 4.

Hence, when an obstacle $\ell_i[u_1, u_2]$ is encountered, two different situations may arise: (i) u_1 and u_2 fall in the same voronoi interval, and (ii) u_1 and u_2 fall in different voronoi intervals. The update of voronoi intervals in both cases is presented in Step 2.2 of algorithm Left-to-Right-Sweep, and is explained in Fig. 6. We now state an important property of our sweep technique in Lemma 5.

LEMMA 5. *When the sweep line $\mathscr{L}$ is moved to the right after encountering a point $p_i \in S$ or an obstacle $\ell_i \in \mathscr{O}$, then before encountering any other point in S or any other obstacle in $\mathscr{O}$, the status of the sweep line, i.e., the voronoi intervals I_j of each $p_j \in S'$ on $\mathscr{L}$, remains unaltered.*

Proof. Referring to Fig. 4, let $[a, b]$ be the voronoi interval I_j of the point $p_j \in S'$ when the sweep line $\mathscr{L}$ encounters p_i. Let the sweep line be advanced by a distance d to the right, so that it does not encounter any other point in S or obstacle in $\mathscr{O}$. Let $[a', b']$ be the projection of $[a, b]$ on the current position of $\mathscr{L}$. We have to prove that $[a', b']$ is also the voronoi interval of p_j at the current position of $\mathscr{L}$.

Let us take a point $m \in [a, b]$ and let m' be its projection on the current position of $\mathscr{L}$. We have to show that $\delta(p_j, m') \leq \delta(p_k, m')$ $\forall p_k \in S'$ $(p_k \neq p_j)$.

On the contrary, let there exist a point $p_k \in S'$ $(p_k \neq p_j)$ such that $\delta(p_k, m') < \delta(p_j, m')$. From Corollary 1, the shortest path P_1 from p_k to m' is monotone in the positive direction of the x axis. Let θ be the point of intersection of the path P_1 and the sweep line $\mathscr{L}$ when it was at p_i. Thus we have $P_1 = \pi(p_k, \theta) \cup \pi(\theta, m')$.

As m is in the voronoi interval I_j, we have $\delta(p_j, m) \leq \delta(p_k, \theta) + \delta(\theta, m)$. Now from Lemma 4, the length of $P_1 = \delta(p_k, \theta) + \delta(\theta, m') =$

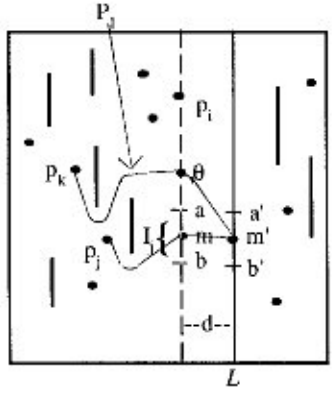


FIG. 4. Proof of Lemma 5.

$\delta(p_k, \theta) + \delta(\theta, m) + \delta(m, m') \geq \delta(p_j, m) + \delta(m, m')$. Hence, m' is nearer to p_j than to p_k. Thus we arrive at a contradiction.

The proof for the case in which the sweep line is moved to the right after encountering an obstacle $\mathscr{O}_i \in \mathscr{O}$ is similar. ∎

Let $C'(\subseteq C)$ be the set of end points already encountered by $\mathscr{L}$ during its left-to-right sweep at the current instant. Let $\mathbb{V} = C \cup S$ and $\mathbb{V}' = C' \cup S'$. Obviously, $\mathbb{V}' \subseteq \mathbb{V}$. We now partition $\mathbb{V}'$ into subsets $\mathbb{V}'_1, \mathbb{V}'_2, \ldots, \mathbb{V}'_{|S'|}$ such that for any element $v_k \in \mathbb{V}'_j$, its left-nearest neighbor is p_j.

During the execution of the algorithm Left-to-Right-Sweep, the voronoi intervals on the sweep line $\mathscr{L}$ are stored in a height balanced tree, and for each voronoi interval I_j, we maintain the shortest path map of p_j. In other words, for I_j we maintain those $v_k \in \mathbb{V}'_j$ that are horizontally visible from I_j (i.e., whose horizontal projection on I_j that are not obstructed by any member in $\mathscr{O}$), sorted by their y coordinates, along with $\delta(v_k, p_j)$'s. The shortest path map, in this context, has been introduced in [5].

Next, we describe our plane sweep algorithm.

4.1.2. *Algorithm Left-to-Right-Sweep*

```
1       sort the elements of V by their x coordinates;
2       for each v_i ∈ V do
2.1       if (v_i ∈ S)        (*sweep line encountered a voronoi site p_j ∈ S
                               (see Fig. 5)*)
2.1.1       determine the voronoi interval I_k of L in which p_j lies;
2.1.2       sequentially search the elements of the shortest path maps
            attached to the voronoi intervals starting from p_j,
            and moving
```

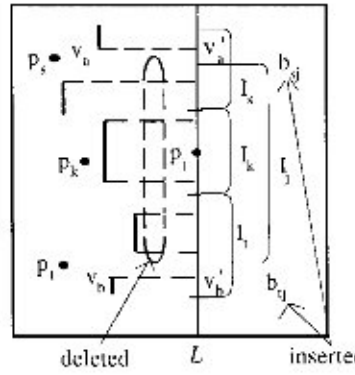


FIG. 5. Processing of voronoi site p_i.

2.1.3.1 upward until we get $v'_a \in I_s[\alpha, \beta]$
(*v'_a *is the horizontal projection of* $v_a (\in \mathbb{V})$ *on* $\mathscr{L}$ and $*I_s[\alpha, \beta]$ is the voronoi interval of p_s in the current position of $\mathscr{L}$*)
such that $\delta(p_s, v_a) + L_1(v_a, v'_a) < L_1(p_j, v'_a)$;
determine a point $b_{s,j} \in I_s$ s.t. $\delta(p_s, v_a) + L_1(v_a, v'_a) + L_1(v'_a, b_{s,j}) = L_1(p_j, b_{s,j})$;

2.1.3.2 downward until we get $v'_b \in I_t[\gamma, \delta]$
(*V_b *is the horizontal projection of* $v_b (\in \mathbb{V})$ *on* $\mathscr{L}$ *and* $*I_t[\gamma, \delta]$ is the voronoi interval of p_t in the current position of $\mathscr{L}$*)
such that $\delta(p_t, v_b) + L_1(v_b, v'_b) < L_1(p_j, v'_b)$;
determine a point $b_{t,j} \in I_t$ s.t. $\delta(p_t, v_b) + L_1(v_b, v'_b) + L_1(v'_b, b_{t,j}) = L_1(p_j, b_{t,j})$

2.1.4 (*update voronoi intervals*)

2.1.4.1 replace $I_s = [\alpha, b_{s,j}]$ in $\mathscr{L}$;
(*voronoi interval for the point p_s*)

2.1.4.2 insert $I_j = [b_{s,j}, b_{t,j}]$ and the point p_j in $\mathscr{L}$;
(*voronoi interval of p_j*)

2.1.4.3 replace $I_t = [b_{t,j}, \delta]$ in $\mathscr{L}$;
(*voronoi interval for the point p_t*)

2.1.4.4 delete all earlier v_j's and $b_{m,n}$'s, that fall in I_j, and insert p_j in I_j;

2.2 if $(v_i \in C)$
(*$\mathscr{L}$ *encountered a line segment obstacle* $\ell_i[u_1, u_2] \in \mathscr{O}$*)

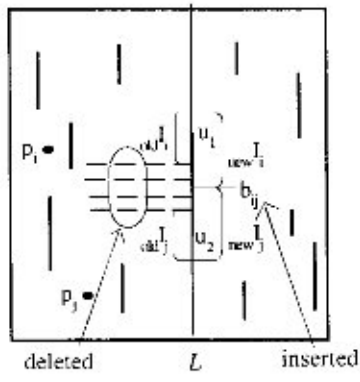


FIG. 6. Processing of line segment $[u_1, u_2]$.

2.2.1 search $\mathscr{L}$ with u_1 and u_2 to find the voronoi intervals $I_i[\alpha_i, \beta_i]$ and $I_j[\alpha_j, \beta_j]$ in which u_1 and u_2 lie, respectively.

(*see Fig. 6*)

2.2.2 if $(I_i = I_j)$ then

2.2.2.1 compute $\delta(u_1, p_i)$ and $\delta(u_2, p_i)$ from the shortest path map attached to I_i;

2.2.2.2 (*insertion of shortest path map of p_i (for details see [5])*)

2.2.2.2.1 insert u_1 (with $\delta(u_1, p_i)$) and u_2 (with $\delta(u_2, p_i)$) in I_i;

2.2.2.2.2 delete all elements of I_i on $\mathscr{L}$ that lie inside the interval $[u_1, u_2]$;

2.2.3 if $(I_i \neq I_j)$ then

2.2.3.1 compute $\delta(u_1, p_i)$ and $\delta(u_2, p_j)$;
(*from the shortest path map attached to I_i and I_j*)

2.2.3.2 let the length of ℓ_i be ℓ;

2.2.3.3 insert u_1 (with $\delta(u_1, p_i)$) and u_2 (with $\delta(u_2, p_j)$) on $\mathscr{L}$;

2.2.3.4 replace $I_i[\alpha_i, \beta_i]$ by $I_i[\alpha_i, b_{i,j}]$, and $I_j[\alpha_j, \beta_j]$ by $I_j[b_{i,j}, \beta_j]$,
(*voronoi intervals for p_i and p_j*)
where $b_{i,j}$ is a point on $\mathscr{L}$ that is at a distance x from u_1, such that $\delta(u_1, p_i) + x = \delta(u_2, p_j) + \ell - x$,
$\Rightarrow x = (\delta(u_2, p_j) - \delta(u_1, p_i) + \ell)/2$,

2.2.3.5 delete all of the earlier v_j's and $b_{m,n}$'s that lie in the interval $[u_1, u_2]$;

endfor

end.

THEOREM 1. *The preprocessing algorithm Left-to-Right-Sweep can be implemented in $O((m + n)\log(m + n))$ time and $O(m + n)$ space.*

Proof. Sorting in Step 1 requires $O((m+n)\log(m+n))$ time. The information on the sweep line $\mathscr{L}$ is maintained in the form of a concatenable queue [2] (using a 2–3 tree). A search for all p_i's and end points of ℓ_i's in Steps 2.1.1 and 2.2.1 can be carried out in $O((m+n)\log(m+n))$ time. The insertions of p_i's (in Step 2.1.4.4), the end points of ℓ_i's (in Steps 2.2.2.1 and 2.2.3.3), and the $b_{m,n}$'s (in Steps 2.1.4.2 and 2.2.3.4) requires $O((m+n)\log(m+n))$ time. The deletion operations are performed in Steps 2.1.4.4, 2.2.2.2.2, and 2.2.3.5, and we observe that once an element is deleted it never reappears in the subsequent stages of the sweep. This fact proves that the total time complexity is $O((m+n)\cdot\log(m+n))$ for the search, insertions, and deletions on $\mathscr{L}$ for all of the elements in $\mathscr{F}$, which is of size $O(m+n)$. The proof of the fact that the space complexity is $O(m+n)$ is quite direct. ∎

In a similar way, using right-to-left sweep, we can compute $RNN(u_i)$ $(=p_j \in S)$ for each end point $u_i \in C$. After two sweep operations for each end point $u_i \in C$, we obtain $LNN(u_i)$ and $RNN(u_i)$, which are p_k and $p_\ell(\in S)$, respectively. Finally, for each end point $u_i \in C$, we maintain $RGVN(u_i)$, which is p_α $(\alpha = k$ or $\ell)$, for which $\delta(p_\alpha, u_i)$ is minimum. Thus we have the following theorem:

THEOREM 2. *Given a set of m points S and a set $\mathscr{E}$ of n vertical line segments, the rectilinear geodesic voronoi neighbor for each endpoint of $\ell_i \in \mathscr{E}$ can be computed in $O((m+n)\log(m+n))$ time in $O(m+n)$ space.*

4.1.3. *Preprocessing Point Sets*

In this section we will discuss the preprocessing of the point set S, so that given an arbitrary isothetic query rectangle $Q[x_I, x_J][y_K, y_L]$, satisfying either $y_K = y_{\min}$ or $y_L = y_{\max}$ and containing a subset $S^*(\in S)$ of points, the rectilinear nearest neighbor of the four corner points of Q, among the members in S^*, can be answered efficiently. Here $y_{\min}$ and $y_{\max}$ are the y coordinates of the top and bottom sides of the bounding rectangle. We shall denote the four corner points of a rectangle by NW, NE, SE, and SW, respectively. In this context, we note that here we will ignore all vertical line segment obstacles in $\mathscr{E}$.

We shall use the *layered segment tree* data structure ($\mathscr{T}$) (Fig. 7) [15] for preprocessing. The points in S are placed in the leaves of $\mathscr{T}$ in sorted order with respect to their x coordinates. The primary structure of $\mathscr{T}$ consists of nodes $v[r:s]$, each corresponding to the set of points $P(v)$ whose x coordinates lie in the interval $[x_{(r)}, x_{(s)}]$, where $x_{(k)}$ is the kth ranked point in the sorted order of points in S by x coordinates. The left and right subtrees are also layered segment trees with the set of points $\{x_{(r)}, \ldots, x_{(\lfloor (r+s)/2 \rfloor)}\}$ and $\{x_{(\lfloor (r+s)/2 \rfloor + 1)}, \ldots, x_{(s)}\}$, respectively. The secondary structure $Y(v)$ attached to a node v is a sorted array of the y

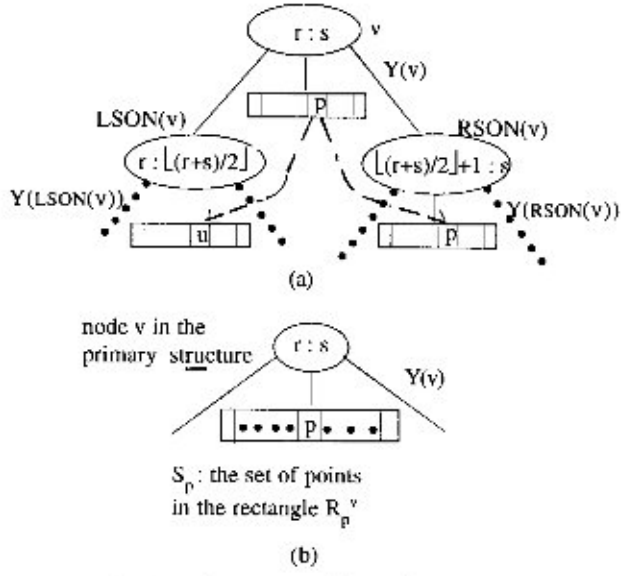


FIG. 7. Node structure of layered segment tree.

coordinates of the points in $P(v)$. It should be noted that each point $p \in Y(v)$ represents a rectangle R_p^v with top left (resp. bottom right) corner $(x_{(r)}, p_y)$ (resp. $(x_{(s)}, y_{\min})$), and it contains all of the points $S_p = \{q \mid q_x \in [x_{(r)}, x_{(s)}] \,\&\, q_y \leq p_y\}$ (see Fig. 7b). We now describe the preprocessing required for answering the $RGVN$ queries for the corners of the rectangle $Q[x_I, x_J][y_K, y_L]$, where $y_L = y_{\min}$. The preprocessing for the case where $y_K = y_{\max}$ is similar.

For efficient searching of the secondary structure during query time, we maintain two additional links for each point p_i in the secondary structure during the preprocessing, as described below.

Note that $LSON(v)$ and $RSON(v)$ are the two children of the node v of the primary structure. So $Y(v)$ is partitioned into two lists $Y(LSON(v))$ and $Y(RSON(v))$, and any element $p \in Y(v)$ will belong to any one of them. Without loss of generality, let $p \in Y(RSON(v))$. Now $p \in Y(v)$ has two pointers, one pointing to the point p in $Y(RSON(v))$ and the other pointing to the point u in $Y(LSON(v))$, such that $u_y = \text{Max}_{t \in Y(LSON(v))}\{t_y \mid t_y \leq p_y\}$ (see Fig. 7a).

Given the query rectangle $Q[x_I : x_J][y_K : y_L]$, with $y_L = y_{\min}$, we perform a binary search at the list Y associated with the root of $\mathcal{T}$ to find a point p such that $p_y \leq y_K$, and the next point p' in the same Y-list has $p'_y > y_K$. In each subsequent Y-list search in the next levels of the primary structure, following the pointers from p, we can determine the point t and

the next point t' such that $t_y \leq y_K$ and $t'_y > y_K$, in constant time. The search in the primary structure with the interval $[x_I : x_J]$ has to consider at most $O(\log m)$ nodes. Each node v where the search stops is of the form $[r : s]$, such that $[x_r : x_s] \subseteq [x_I : x_J]$. Note that, for a node $[r : s]$, the property that $[x_r : x_s] \subseteq [x_I : x_J]$, is valid for all of its successor nodes, so they need not be considered further. In the secondary structure (Y-list) of each of these nodes, we identify the point t and the next member t' satisfying $t_y \leq y_K$ and $t'_y > y_K$. The nearest neighbor n_v from the NW corner of R_t^v can be found from the preprocessed data structure. From $O(\log m)$ nodes of the primary structure, we get $O(\log m)$ n_v's. Finally, the answer to the query will be the point among these $O(\log m)$ n_v's, which is at the minimum distance from the NW corner of the rectangle Q.

Regarding the space complexity, the primary structure of the layered segment tree $\mathcal{T}$ is a height balanced tree of depth $O(\log m)$, and it requires $O(m)$ space. Each point $p \in S$ appears in the secondary structure of exactly one node in all of the levels of the primary structure of $\mathcal{T}$, and each of the occurences consumes $O(1)$ space. This implies that the total space required to store the secondary structures is $O(m \log m)$. For this data structure, the preprocessing and query time complexities are $O(m \log m)$ and $O(\log m)$, respectively [12, 15, 17]. Thus we have established the following theorem:

THEOREM 3. *A set S of m points can be preprocessed in $O(m \log m)$ time and with $O(m \log m)$ storage, using a layered segment tree, such that for a given query rectangle $Q[x_I : x_J][y_K : y_L]$, with either $y_K = y_{\max}$ or $y_L = y_{\min}$ and containing a subset $S^*(\in S)$ of points in its interior, it is possible to report the geodesic voronoi neighbor $p \in S^*$ from any of the four corner points, in $O(\log m)$ time.*

4.2. *Query Answering*

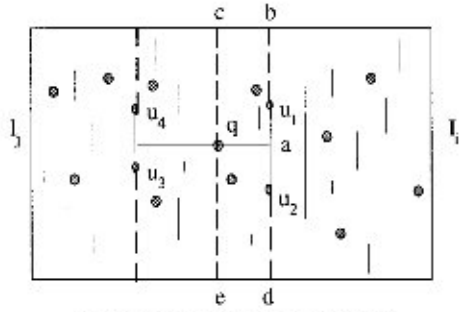
Given an arbitrary query point q ($q \notin S$), we now discuss the method of finding its rectilinear geodesic voronoi neighbor among the members of the set S from the preprocessed data structure. We shall describe an algorithm for finding the right nearest neighbor $RNN(q)$, for an arbitrary query point q, in the $+X$ direction.

4.2.1. *Algorithm RNN-Query*

Step 1 Shoot a ray from q to the right. The following two cases may arise:

(i) The ray hits a vertical line segment obstacle $\ell_i(u_1, u_2) \in \mathcal{O}$ (see Fig. 8a).

(ii) The ray hits the right boundary of the bounding rectangle (see Fig. 8b).



(a) when ray hits an obstacle

(b) when ray hits the floor boundary

FIG. 8. Query processing.

Step 2 For case (i), find $p_k = RGVN(u_1)$ and $p_\ell = RGVN(u_2)$, with rectilinear distances $\delta(u_1, p_k)$ and $\delta(u_2, p_\ell)$, respectively. Let $\delta(q, p_k) = L_1(q, u_1) + \delta(u_1, p_k)$ and $\delta(q, p_\ell) = L_1(q, u_2) + \delta(u_2, p_\ell)$.

Step 3 For both cases (i) and (ii), find the geodesic nearest neighbors of q inside the rectangles $\square qabc$ and $\square qade$ from the layered segment tree $\mathcal{T}$. Let these be p_i and p_j, respectively. Compute $\delta(q, p_i) = L_1(q, p_i)$ and $\delta(q, p_j) = L_1(q, p_j)$.

Step 4 For case (i), $RNN(q) = p_\alpha$ if $\delta(q, p_\alpha) = \text{Min}(\delta(q, p_\beta), \beta = i, j, k, \ell)$. For case (ii), $RNN(q) = p_\alpha$ if $\delta(q, p_\alpha) = \text{Min}(\delta(q, p_\beta), \beta = i, j)$.

A similar procedure, *LNN*-Query determines $LNN(q)$ for the query point q. Finally, $RGVN(q)$ can be found out by using Eq. (1) of Section 4.1.1.

LEMMA 6. *Algorithm RNN-Query runs in* $O(\log(m + n))$ *time.*

Proof. To perform the ray shooting query in Step 1, a point location query is to be solved over the horizontal trapezoidation of $\mathcal{O}$, and the rectangle in which q lies is to be identified. This requires $O(\log n)$ time, using the algorithm of Kirkpatrick [9]. Step 2 can be carried out in $O(1)$ time. Step 3 can be performed in $O(\log m)$ time in the worst case. This follows directly from Theorem 3. Thus the time complexity of the query algorithm is $O(\log(m + n))$. ∎

The *layered segment tree*, constructed with the voronoi sites in S, consumes $O(m \log m)$ space (by Theorem 3). The spaces required for computing and storing the rectilinear geodesic voronoi neighbors for the end points of the vertical line segment obstacles in $\mathcal{O}$ are $\mathcal{O}(m + n)$ and $O(n)$, respectively. Thus the total space complexity of our preprocessing algorithm is $O(m \log m + n)$.

Thus Theorem 2, Lemma 6, and the above discussion lead to the following theorem:

THEOREM 4. *Given a set S of m points and a set $\mathcal{O}$ of n vertical line segment obstacles on a rectangular floor, the rectilinear geodesic voronoi neighbor of an arbitrary query point q can be computed in $O(\log(m + n))$ time by using $O((m + n)\log(m + n))$ preprocessing time. The space required to store the data structure is $O(m \log m + n)$.*

5. RGVN QUERIES AMONG RECTANGULAR OBSTACLES

In this section we consider the problem of computing the rectilinear geodesic nearest neighbor of an arbitrary query point q, among a set S of m points located on the periphery of the rectangular floor in the presence of a set $\mathcal{O}$ of n disjoint isothetic (i.e., whose sides are parallel to the enclosing box) rectangular obstacles inside the floor. Let A_1, A_2, A_3, and A_4 be the sets of points on the left, top, right, and bottom sides of the floor, respectively. Thus $S = A_1 \cup A_2 \cup A_3 \cup A_4$. The rectilinear geodesic nearest neighbor of an arbitrary query point q among the set of points in A_1, A_2, A_3, and A_4 will be denoted by $p_{-x}(q)$, $p_{+y}(q)$, $p_{+x}(q)$, and $p_{-y}(q)$, respectively. Next, we explain the method of finding $p_{-x}(q)$.

From the query point q, we shoot a ray to the left. It either hits the right side of an obstacle (say, $R_i \in \mathcal{O}$) or the left boundary of the floor. Now let us consider the following two lemmas.

LEMMA 7. *If the ray hits the left boundary of the floor at a point t, then $p_{-x}(q)$ will be the point of S placed in A_1 that is either just vertically above or is just vertically below t.*

The proof of the lemma follows directly from the definition of $p_{-x}(q)$.

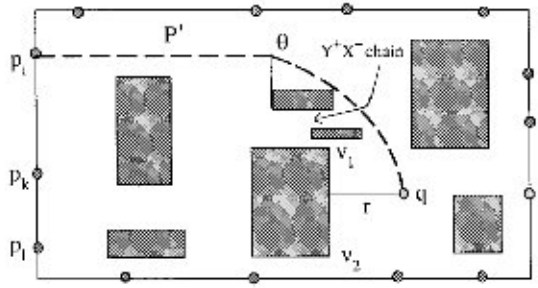


FIG. 9. Proof of Lemma 8.

LEMMA 8. *If the ray hits the right vertical edge* (v_1, v_2) *of a rectangular obstacle* $R_i \in \mathscr{O}$, *then* $p_{-x}(q)$ *will be either* $p_{-x}(v_1)$ *or* $p_{-x}(v_2)$.

Proof. [By contradiction]. Let us assume $p_{-x}(q) = p_i$ $(p_i \in A_1)$. Let p_k and p_ℓ be two points in A_1 such that $p_{-x}(v_1) = p_k$ and $p_{-x}(v_2) = p_\ell$. Note that $\delta(p_i, q) < \min(\delta(p_k, q), \delta(p_\ell, q))$. Now the shortest path P' $(= \pi(p_i, q))$ will be one of the following three types:

Case I	The path is monotone in the X^+Y^- direction.
Case II	The path is monotone in the X^+Y^+ direction.
Case III	The path is monotone only in the X^+ direction.

In Case I (see Fig. 9), consider a Y^+X^- chain from v_1 that intersects $\pi(p_i, q)$ at the point θ. Now we have

$$L_1(v_1, \theta) + \delta(\theta, p_i) = \delta(v_1, p_i) \tag{1}$$

since both of the paths are monotone in both directions, or

$$L_1(q, v_1) + L_1(v_1, \theta) + \delta(\theta, p_i) = \delta(q, p_i) \tag{2}$$

Note that

$$\delta(v_1, p_k) \leq \delta(v_1, p_i) \quad (\text{as } p_k = p_{-x}(v_1)). \tag{3}$$

Again,

$$\begin{aligned} \delta(q, p_k) &\leq L_1(q, v_1) + \delta(v_1, p_k) \\ &\leq L_1(q, v_1) + \delta(v_1, p_i) && \text{(From (3))} \\ &= L_1(q, v_1) + L_1(v_1, \theta) + \delta(\theta, p_i) && \text{(From (1))} \\ &= \delta(q, p_i) && \text{(From (2))} \end{aligned}$$

Thus $\delta(q, p_k) \leq \delta(q, p_i)$, which leads to a contradiction.

In Case II, using similar arguments, we can show that $\delta(q, p_\ell) \leq \delta(q, p_i)$.

In Case III, we know (from Lemma 2) that there exists a shortest path from p_i to q that passes through either v_1 or v_2. Without loss of generality, let us assume it passes through v_1. The argument when it passes through v_2 is similar. We now have

$$\begin{aligned}\delta(q, p_i) &= L_1(q, v_1) + \delta(v_1, p_i) \\ &\geq L_1(q, v_1) + \delta(v_1, p_k) \quad (\text{since } p_k = p_{-x}(v_1)) \\ &\geq \delta(q, p_k).\end{aligned}$$

Thus $\delta(q, p_i) \geq \delta(q, p_k)$, which leads to a contradiction, as we assumed $p_i = p_{-x}(v_1)$. ∎

We now introduce a data structure, called a *carrier graph* [6], which will be used to maintain the monotone chains from each point $p_i \in S$. There are four types of carrier graphs, G_{+X}, G_{-X}, G_{+Y}, and G_{-Y}. Each graph is a directed acyclic planar graph with nonnegative edge weights, and it contains paths that are monotonic with respect to a specified direction. These graphs exploit the knowledge that the shortest path between any two points avoiding the obstacles in $\mathscr{O}$ is monotonic in at least one of the X or Y directions [5]. Each of the four carrier graphs has $O(n)$ vertices and edges, where n is the number of rectangles in $\mathscr{O}$, and they can be computed in $O(n \log n)$ time. These graphs contain sufficient information to support shortest path queries.

For each point $p_i \in A_1$, both the Y^+X^+ and Y^-X^+ chains and the shortest path to each corner point in C obtained from $\mathscr{O} \cup A_1$ are maintained in G_{+X} [6].

From the carrier graph, we shall construct four data structures, Vor_{+X}, Vor_{+Y}, Vor_{-X}, and Vor_{-Y}, which will be used for reporting the voronoi neighbor queries for the corner points of the obstacles in $\mathscr{O}$. We now describe the construction of Vor_{+X} based on Lemmas 9 and 10. The constructions of Vor_{+Y}, Vor_{-X}, and Vor_{-Y} are similar.

LEMMA 9 [8]. *In a directed acyclic graph $G = (V, E)$, the shortest path tree from a source vertex s can be computed in $O(|V| + |E|)$ time.*

LEMMA 10 [13]. *Given a directed acyclic graph $G = (V, E)$ with nonnegative edge weights and $\mathbb{V} \subset V$, where $|\mathbb{V}| = k$, we can partition V into subsets $V_1, V_2, \ldots, V_k$ such that $\bigcup_{i=1}^{k} V_i = V$ and $v_i \in \mathbb{V}$ is nearer to all vertices of V_i, compared to all other vertices $v_j \in \mathbb{V}$, $j \neq i$. This partition can be computed in $O(|V| + |E|)$ time.*

5.1. *Algorithm Build_Vor$_{+X}$*

Step 1 Compute the carrier graph G_{+X} on $\mathscr{O} \cup A_j$ by applying a plane sweep as described in [6].

Step 2 Apply Mehlhorn's algorithm [13] as described in Lemma 10 on G_{+X}, with V = the set of vertices in A_1. Subsequently, with each corner point of the rectangles in $\mathscr{O}$, we associate the point $p \in A_1$ that is found to be geodesically closest in this step, and the shortest distance of that corner point from p.

Besides maintaining Vor$_{+X}$, Vor$_{+Y}$, Vor$_{-X}$, and Vor$_{-Y}$, we also maintain two additional structures corresponding to the horizontal and vertical trapezoidation of the free space avoiding the obstacles in $\mathscr{O}$. These will be used to answer the ray shooting queries [9]. We also maintain four sorted lists for the points in A_1, A_3 (sorted with respect to their y-coordinates) and A_2, A_4 (sorted with respect to their x coordinates).

THEOREM 5. *The time and space complexities of the preprocessing step are* $O((m+n)\log(m+n))$ *and* $O(m+n)$, *respectively.*

Proof. Consider the construction of Vor$_{+X}$. Construction of carrier graphs G_{+X} (on $\mathscr{O} \cup A_1$) requires $O((m+n)\log(m+n))$ time [6]. Step 2 of Build_Vor$_{+X}$ can be carried out in $O(m+n)$ time, since the graph G_{+X} is a planar graph (using Lemma 10). The constructions of Vor$_{+Y}$, Vor$_{-X}$, and Vor$_{-Y}$ are similar and can be carried out in $O((m+n)\cdot \log(m+n))$ time in the worst case.

The horizontal and vertical trapezoidations of $\mathscr{O}$, maintained during the preprocessing for *point location query*, require $O(n \log n)$ time. The sorted lists of the points in A_1, A_2, A_3, and A_4 are also obtained in $O(m \log m)$ time. Thus the overall time complexity of preprocessing is $O((m+n)\cdot \log(m+n))$, and all of the data structures can be maintained in $O(m+n)$ space. ∎

5.2. *Answering the RGVN Queries*

Now we describe an algorithm for answering the *RGVN* queries for an arbitrary query point $q \in S$.

5.2.1. *Algorithm Geodesic_Neighbor_A$_1$ (*RGVN(q) in A_1 is $p_{-x}(q)$*)*

Step 1 Shoot a horizontal ray to the left from the point q. Let it hit the vertical edge u_iu_j in the horizontal trapezoidation. If u_iu_j is a part of a right vertical edge of an obstacle in $\mathscr{O}$, then go to Step 2; else go to Step 3.

Step 2 Let v_iv_j be the edge of the obstacle whose part is u_iu_j. Let $p_k = RGVN(v_i)$ and $p_\ell = RGVN(v_j)$, respectively, obtained from Vor_{+X}. If $L_1(q, v_i) + \delta(v_i, p_k) < L_1(q, v_j) + \delta(v_j, p_\ell)$, then $p_{-x}(q) = p_k$; else $p_{-x}(q) = p_\ell$. Stop.

Step 3 (*u_iu_j *is a part of the left side of the boundary rectangle* Z*) Perform a binary search in A_1 to find p_k and p_ℓ such that $p_{ky} \le q_y \le p_{\ell y}$. If $L_1(p_k, q) < L_1(p_\ell, q)$, then $p_{-x} = p_k$; else $p_{-x} = p_\ell$. Stop.

In a similar way, using Vor_{+Y}, Vor_{-X}, and Vor_{-Y}, we determine p_{-y}, p_{+x}, and p_{-y} respectively. Finally, $RGVN(q) = p_{id}(q)$, where $id \in \{+x, -x, +y, -y\}$, for which $\delta(q, p_{id}(q))$ is a minimum. Thus, we have the final theorem:

THEOREM 6. *The RGVN query for an arbitrary query point q can be answered in $O(\log(m + n))$ time.*

Proof. The time required for ray shooting in Step 1 of the algorithm Geodesic_Neighbor_A_1 is $O(\log n)$, since it requires solving a point location query over the horizontal trapezoidation of $\mathscr{E}$. Once we identify the rectangle of the trapezoidation in which q belongs, the remaining part of the computation in this step requires $O(1)$ time. Step 2 requires $O(1)$ time. If Step 3 is executed, the binary search takes $O(\log m)$ time. Thus the worst case time complexity of the algorithm is bounded by $O(\log(m + n))$. ∎

Thus from Theorems 5 and 6, we have established the following theorem, which is the main result of this paper.

THEOREM 7. *The rectilinear geodesic voronoi neighbor of an arbitrary query point q, denoted by $RGVN(q)$, can be answered in $O(\log(m + n))$ time in the worst case by using a preprocessing step that requires $O((m + n) \cdot \log(m + n))$ time in the worst case. The space complexity of the algorithm is $O(m + n)$.*

5.3. *Parallelizing the Preprocessing Step*

Parallelization of the preprocessing step is quite direct from the following results.

LEMMA 11. *Preprocessing step of our algorithm can be implemented in parallel with $O((m + n)^{1.5}/\log^2(m + n))$ processors and in $O(\log^3(m + n))$ time.*

Proof. Step 1 of Build_Vor_{+X} can be easily parallelized by using the *parallel plane sweep* technique of Atallah et al. [1] using $O(m + n)$ processors and in $O(\log(m + n))$ time. Step 2 can be parallelized by using the

parallel single source shortest path algorithm of Pan and Rief [16], which works on a planar graph in $O(\log^3(m+n))$ time, using $O((m+n)^{1.5}/\log^2(m+n))$ processors.

The preprocessing to answer the planar point location query can be parallelized by using the algorithm of Dadoun and Kirkpatrick [4], using $O(m+n)$ processors and $O(\log(m+n))$ time. Furthermore, sorting of four lists A_1, A_2, A_3, and A_4 can be parallelized by using $O(m)$ processors in $O(\log m)$ time [3]. Thus the overall processor and time complexities are bounded by $O((m+n)^{1.5}/\log^2(m+n))$ and $O(\log^3(m+n))$, respectively. ∎

Thus we have established the following theorem:

THEOREM 8. *The geodesic voronoi neighbor of an arbitrary query point q can be computed in $O(\log(m+n))$ time after a preprocessing that can be done in parallel by using $O((m+n)^{1.5}/\log^2(m+n))$ processors and in $O(\log^3(m+n))$ time.*

6. CONCLUSION AND OPEN PROBLEMS

This paper presents two simple algorithms for finding the rectilinear geodesic voronoi neighbor (site) of an arbitrary query point q, where n sites are located on a rectangular region among a set of m isothetic obstacles.

The first one works in $O(\log(m+n))$ time, where the obstacles are vertical line segments. This requires a preprocessing step that takes $O((m+n)\log(m+n))$ time, using a layered segment tree data structure, and stores the preprocessed information in a data structure of size $O(m \log m + n)$. This is an improvement over previously known results [7, 14].

In the second algorithm, the obstacles may be isothetic rectangles, but the sites are located on the periphery of the bounding rectangle. This algorithm uses a planar graph, called the carrier graph, for preprocessing and solves the geodesic voronoi neighbor query in $O(\log(m+n))$ time. The preprocessing time for this problem is $O((m+n)\log(m+n))$, and it consumes $O(m+n)$ space.

The possible directions of extending the work are

(i) Reducing the space complexity of our first problem to $O(m+n)$.

(ii) Considering the more general problem, in which the obstacles are disjoint arbitrary polygons and sites are placed inside the bounding box, so that the $RGVN$ query for any arbitrary point can be answered in $O(\log(m+n))$ time with a preprocessing time $O((m+n)\log(m+n))$ using $O(m+n)$ space.

(iii) Finally, designing an efficient parallel algorithm for the above-mentioned general problem.

ACKNOWLEDGMENTS

A preliminary version of this paper has appeared in the proceedings of the 16th annual conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS-16), December 1996 (LNCS 1180), Springer-Verlag, Berlin. We thank both the referees for their suggestions to improve the quality of the paper.

REFERENCES

1. M. J. Attallah, R. Cole and M. T. Goodrich, Cascading Divide-and-Conquer: A technique for designing parallel algorithms, *in* "Proceedings of the Symposium on the Foundations of Computer Science," 1987, pp. 151–160.
2. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, "Design and Analysis of Computer Algorithms," Addison-Wesley, Reading, MA, 1974.
3. R. Cole, Parallel merge sort, *SIAM J. Comput.* **17** (1988), 770–785.
4. N. Dadoun and D. Kirkpatrick, Parallel construction of subdivision hierarchies, *J. Comput. System Sci.* **39** (1989), 153–165.
5. P. J. de Rezende, D. T. Lee, and Y. F. Wu, Rectilinear shortest paths with rectangular barrier, *Discrete Comput. Geom.* **4** (1989), 41–53.
6. H. Elgindy and P. Mitra, Orthogonal shortest path queries among axes parallel rectangular obstacles, *Internat. J. Comput. Geom. Appl.* **4**(1) (1994), 3–24.
7. S. Guha and I. Suzuki, Proximity problems for points on rectilinear plane with rectangular obstacles, *Algorithmica*, **17** (1997), 281–307.
8. D. B. Johnson, Efficient algorithms for shortest paths in sparse network, *J. Assoc. Comput. Mach.* (1977), 1–13.
9. D. G. Kirkpatrick, Optimal search in planar subdivision, *SIAM J. Comput.* **12** (1983), 28–35.
10. R. C. Larson and V. O. K. Li, Finding minimum rectilinear distance paths in the presence of barriers, *Networks* **11** (1981), 285–304.
11. D. T. Lee and F. P. Preparata, Euclidean shortest path among rectilinear barrier, *Networks* (1983), 393–410.
12. G. S. Lueker, A data structure for orthogonal range queries, *in* "Proceedings of the 19th Annual IEEE Symposium on the Foundations of Computer Science," 1978, pp. 28–34.
13. K. Mehlhorn, A faster approximation algorithm for the Steiner problems in graphs, *Inform. Process. Lett.* **27** (1988), 125–128.
14. J. S. B. Mitchell, L_1 shortest paths among polygonal obstacles in the plane, *Algorithmica* **8** (1992), 55–88.
15. F. P. Preparata and M. I. Shamos, "Computational Geometry—An Introduction," Springer-Verlag, New York, 1985.
16. V. Pan and J. Reif, Fast and efficient solution of path algebra problems, *J. Comput. System Sci.* **38** (1989), 494–510.
17. D. E. Willard, "Predicate-Oriented Database Search Algorithms," Ph.D. Thesis, Report TR-20-78, Aiken Corporation Laboratory, Harvard University, Cambridge, MA, 1978.