An Algorithm for Finding a Region with the Minimum Total L_1 from Prescribed Terminals

Yoshiyuki Kusakari[1] and Takao Nishizeki[2]
Graduate School of Information Sciences
Tohoku University
Sendai 980-77, Japan

Abstract Given k terminals and n axis-parallel rectangular obstacles on the plane, our algorithm finds a plane region R^* such that, for any point p in R^*, the total length of the k shortest rectilinear paths connecting p and the k terminals without passing through any obstacle is minimum. The algorithm is output-sensitive, and takes $O((K+n)\log n)$ time and $O(K+n)$ space if k is a fixed constant, where K is the total number of polygonal vertices of the found region R^*.

1 Introduction

For k terminals and n axis-parallel rectangular obstacles on the plane, the *optimal region* R^* is a plane region such that, for any point p in R^*, the total length of the k shortest rectilinear paths connecting p and the k terminals without passing through any obstacle is minimum. The optimal region R^* is not always connected, but consists of one or more connected polygons. We denote by K the total number of vertices of these polygons. Thus K is the size of a polygonal representation of R^*. Although K is $O(k^2n^2)$, K is often very small. In this paper, we give an efficient algorithm to find such an optimal region for given terminals and obstacles. The algorithm is output-sensitive, and takes $O((K+n)\log n)$ time and $O(K+n)$ space if k is a fixed constant.

The problem of finding a region with the minimum total L_1 distance from some prescribed sites often appears in many practical problems [4, 13]. For example, when a tenant decides which apartment house he rents, he may wish to minimize the total distance from the apartment house to some prescribed sites, say a school, a railway station, a post office, a hospital etc. If all roads are "axis-parallel" like in Manhattan, the total distance should be measured by the L_1 distance.

A similar problem appears also in the design of multi-layer VLSI layouts [14, 15]. Two rectilinear wires which connect pairs of "pins" and cross on the same layer must change their layers at a "via" to prevent an electric short circuit, and the total length of the two wires should be minimized. In this problem, the "via" should be put in a region such that the total L_1 distance from a point in the region to the four pins is minimum.

Our algorithm can be applied to these problems. In the reference [11], Kusakari *et al.* presented an algorithm for finding a pair of rectilinear paths connecting

[1] kusakari@nishizeki.ecei.tohoku.ac.jp
[2] nishi@ecei.tohoku.ac.jp

$k = 4$ terminals, which neither pass through any rectangular obstacles, nor cross each other except in "crossing areas," and the sum of lengths of which is minimum. In this paper we use the techniques developed in [11], extending them for general k. Guha and Suzuki presented efficient algorithms for some proximity problems on a rectilinear plane with rectangular obstacles, but their problems and techniques are different from ours [5].

The idea behind our algorithm is as follows: we first find all polygonal vertices of the optimal region R^* by plane sweep using a sparse graph on which all vertices lie, and then, connecting them, we find all polygonal edges of R^*.

2 Preliminaries

In this section we first define several terms and problems. As a preprocessing, our algorithm divides the plane into four subregions for each terminal. We then show how to divide the plane, and finally present a known result for the division.

The x-coordinate of a point $p \in \mathbb{R}^2$ is denoted by $x(p)$, and the y-coordinate by $y(p)$. The point p is often denoted by $(x(p), y(p))$. The closed line-segment connecting two points $p_1, p_2 \in \mathbb{R}^2$ is denoted by $[p_1 - p_2]$. A horizontal or vertical line segment is called an *axis-parallel* line segment. In this paper we consider only *rectilinear* paths, that is, those consisting of axis-parallel line segments. All rectangles are assumed *axis-parallel*, that is, all edges of them are axis-parallel. The *length* of a rectilinear path $P \subseteq \mathbb{R}^2$ is the sum of lengths of line segments in P.

We assume that there are n rectangular obstacles on the plane $\mathbb{R}^2$ which do not overlap each other. The set of obstacles is denoted by $\mathcal{O} = \{O_1, O_2, \cdots, O_n\}$. The boundary of a plane region $Q \subseteq \mathbb{R}^2$ is denoted by $B(Q)$. The *routing region* A is a subregion of $\mathbb{R}^2$ excluding the proper insides of obstacles, that is, $A = \mathbb{R}^2 - \bigcup\{O_i - B(O_i) | O_i \in \mathcal{O}\}$. Thus the routing region A includes the boundaries of obstacles, and hence paths can pass through a boundary of two touching obstacles. The left-upper, left-below, right-upper, right-below vertices of a rectangle $Q \subseteq \mathbb{R}^2$ are denoted by $lu(Q)$, $lb(Q)$, $ru(Q)$ and $rb(Q)$, respectively. The upper and below edges of Q are denoted by $ue(Q)$ and $be(Q)$, respectively. We also assume that there are k terminals $t_1, t_2, \cdots, t_k$ in A, and that k is a fixed constant. The set of terminals is denoted by $T = \{t_1, t_2, \cdots, t_k\}$.

The *distance* in A between p_1 and p_2 is defined to be the length of a shortest path connecting p_1 and p_2 in A, and denoted by $d(p_1, p_2)$. For any point $p \in A$, we denote by $d_T(p) = \sum_{t_i \in T} d(p, t_i)$ the *total distance* of p to the terminals. Let $d^* = \min_{p \in A} d_T(p)$ be the *minimum total distance*. The region $R^* = \{p \in A | d_T(p) = d^*\}$ is called the *optimal region*. For a positive real number c, we define the *feasible region* $R(c)$ as $R(c) = \{p \in A | d_T(p) \leq c\}$.

In this paper, we present an efficient algorithm for finding an optimal region R^* for given two sets $\mathcal{O}$ and T. The algorithm is output-sensitive, and finds the optimal region R^* in time $O((K + n) \log n)$ and in space $O(K + n)$, where K is the number of vertices of the optimal region R^*. The number K is often very small. We indeed present an algorithm to find a feasible region $R(c)$. Slightly modifying the algorithm, one can immediately obtain an algorithm to compute the minimum total distance d^*. An optimal region R^* is merely a feasible region $R(d^*)$.

A point w on the boundary $B(R(c))$ of $R(c)$ is called a *proper vertex of* $R(c)$ if w is a polygonal vertex of $B(R(c))$. On the other hand, w is called a *degenerated vertex of* $R(c)$ if w is not a proper vertex of $R(c)$ but $R(c+\varepsilon)$ has a proper vertex in a neighborhood of w for any small positive number $\varepsilon > 0$. A point $w \in B(R(c))$ is called a *vertex of* $R(c)$ if w is a proper or degenerated vertex of $R(c)$. The set of all vertices of $R(c)$ is denoted by $V(R(c))$. Thus $K = |V(R(c))|$.

Let $L = [p_1 - p_1']$ be a vertical line segment in A. We say that a point p in A is *visible from* L *in the* x*-direction* if $y(p_1) \leq y(p) \leq y(p_1')$, $x(p_1) \leq x(p)$, and the horizontal line segment from p to L intersects none of the obstacles. We similarly define a *point visible from* L *in the* $(-x)$*-direction* and a *point visible from a horizontal line segment in the* $(\pm y)$*-direction.*

There are several algorithms to find single-source shortest paths in a routing region [1, 2, 3, 7, 8, 9]. An algorithm by de Rezende *et al.* [3] divides the routing region A into four subregions [3]. Our algorithm uses the same division of A, which we explain below.

A path P in A is defined to be *monotone in the* x*-direction* if the intersection of P and any vertical line is either a single point or a single line segment. Such a path is called an x*-path.* Similarly we define $(-x)$-, y-, and $(-y)$*-paths.*

We recursively define an xy*-path* $P \subseteq A$ starting at a point $t \in A$ as follows [3]:

(1) P is the semi-infinite horizontal line $y = y(t)$ for $x \geq x(t)$, and does not intersect any obstacle; or

(2) P follows the $+x$-direction to a vertical edge of an obstacle, then follows the $+y$-direction up to the upper end p of the edge, and then follows an xy-path starting at p (See Fig. 1).

We similarly define $x(-y)$-, $-xy$-, and $-x(-y)$*-paths.* These four paths, illustrated in Fig. 1, divide A into four regions A_{tx}, $A_{t(-x)}$, A_{ty}, $A_{t(-y)}$. For a terminal $t \in T$, the x*-region* A_{tx} is the subregion of A which is bounded by the xy- and $x(-y)$-paths starting at t and contains the $+x$-semi-axis. We similarly define the $(-x)$*-region* $A_{t(-x)}$, y*-region* A_{ty}, and $(-y)$*-region* $A_{t(-y)}$, as illustrated in Fig. 1. We similarly define a division of $\mathbb{R}^2$ into four subregions ${\mathbb{R}^2}_{tx}$, ${\mathbb{R}^2}_{t(-x)}$, ${\mathbb{R}^2}_{ty}$ and ${\mathbb{R}^2}_{t(-y)}$. One can easily observe that the following lemma holds [3].

Lemma 2.1 *Let* $t \in T$ *and* $z \in \{\pm x, \pm y\}$. *Then, for any point* p *in* A_{tz}, *there exists a shortest path connecting* t *and* p *in* A *which is a* z*-path and is contained in* A_{tz}. □

One can find the four paths for each terminal $t \in T$ in time $O(n \log n)$ using $O(n)$ space [3]. Since $k = O(1)$, the subdivisions of the routing region A for all terminals $t \in T$ can be found in time $O(n \log n)$.

Our algorithm computes the distance $d(t,p)$ between a terminal $t \in T$ and any point $p \in A_{tz}$ by a plane sweep method. For a horizontal line segment L

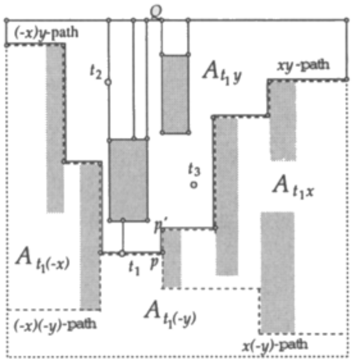

Fig. 1. Plane graph $G_{t_1 y}$.

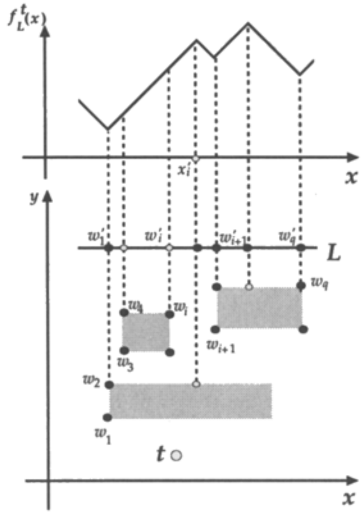

Fig. 2. Illustration for $f_L^t(x)$.

in A, a function $f_L^t : \{x|p = (x, y) \in L\} \to \mathbb{R}$ is defined as follows: $f_L^t(x) = d(t,p)$. A function $f_L^t(y)$ is similarly defined for a vertical line segment L. The function $f_L^t(x)$ is piecewise linear in x, and the slope is $+1$ or -1 as illustrated in Fig. 2. Therefore the function f_L^t is represented by a sequence of continuous line segments with ± 1 slopes. We often denote simply by f_L^t the sequence of line segments representing f_L^t.

Let $t \in T$, $z \in \{\pm x, \pm y\}$, and let L_h be a horizontal line segment in A_{tz}. A point on L_h at which the slope of line segments representing $f_{L_h}^t$ changes the value is called a *bend point of* $f_{L_h}^t$ or simply a *bend point on* L_h. If an end point of L_h is on a vertical edge of a rectangular obstacle, then the end point is also defined to be a *bend point*. We similarly define the bend point on vertical line segment L_v. Thus $\frac{\partial}{\partial x} d(t,p)$ is discontinuous at a bend point on a horizontal line segment L_h, and $\frac{\partial}{\partial y} d(t,p)$ is discontinuous at a bend point on a vertical line segment L_v. For a horizontal or vertical line segment L, the function f_L^t can be represented by the ends of L, the bend points on L, and the slopes between two consecutive ones among them. The following lemma holds on the total number of bend points on all edges of obstacles in A_{tz} [11, 12].

Lemma 2.2 *Let $t \in T$ and $z \in \{\pm x, \pm y\}$. Then there are at most $8n + 1$ points which are bend points on edges of obstacles in A_{tz} or are vertices of obstacles.*

□

We define a function f_L^T as follows. For a horizontal line segment L in A, a function $f_L^T : \{x|p = (x,y) \in L\} \to \mathbb{R}$ is defined as $f_L^T(x) = d_T(p)$. We similarly define $f_L^T(y)$ for a vertical line segment L. A point p on L is called a *bend point* of f_L^T if there is a terminal $t \in T$ such that p is a bend point of f_L^t. Hence every point $p \in L$ at which the slope of line segments representing $f_{L_h}^T$ changes the value, that is, at which $\frac{\partial}{\partial x} d_T(p)$ or $\frac{\partial}{\partial y} d_T(p)$ is discontinuous, is a bend point of $f_{L_h}^T$.

3 Algorithm for finding a feasible region

In this section we present an algorithm **Feasible-Region** to find a feasible region $R(c)$ in $O((K+n)\log n)$ time using $O(K+n)$ space. The outline of the algorithm is as follows.

Algorithm Feasible-Region
begin
1. Construct a graph G_t for each $t \in T$;
2. Construct a graph G_T from the k graphs G_t, $t \in T$, by taking their union;
3. Find the set $V(R(c))$ of vertices of $R(c)$ by plane sweep in the x- and y-directions using graph G_T;
4. Find the boundary $B(R(c))$ by connecting some of the pairs of vertices in $V(R(c))$ as polygonal edges
end.

In Section 3.1 we present an algorithm to construct k plane graphs G_t. In Section 3.2 we construct a nonplanar graph G_T. In Section 3.3, using graph G_T, we find all vertices in $V(R(c))$. In Section 3.4, connecting some of the pair of vertices in $V(R(c))$, we find the boundary $B(R(c))$.

3.1 Graph G_t

In this section we present an algorithm to construct graph G_t for each terminal $t \in T$. For the purpose, we construct four plane graphs G_{tz}, $z \in \{\pm x, \pm y\}$. Since the construction of these four graphs are similar, we show how to construct G_{ty}. Graph $G_{t_1 y}$ is illustrated in Fig. 1. For convenience's sake, we consider a sufficiently large rectangle $Q^\infty \subseteq \mathbb{R}^2$. One can assume without loss of generality that all terminals, all obstacles and $R(c)$ are in the proper inside of Q^∞. Add first to G_{ty} terminal t and all corners of boundary $B({\mathbb{R}^2}_{ty} \cap Q^\infty)$ as vertices. Add next to G_{ty}, as edges, line segments connecting any two consecutive vertices which are on the boundary $B({\mathbb{R}^2}_{ty} \cap Q^\infty)$ and are not on the upper edge of Q^∞. We add more vertices and edges to G_{ty} during plane sweep as follows.

We move a horizontal sweep line L in the $+y$-direction from $y(t)$ to the upper edge of Q^∞ with stopping on each of the horizontal edges of obstacles in A_{ty}. We keep data on L, and update the data whenever L stops. We use a *segment tree* T_t as the data structure [16]. $L \cap A_{ty}$ contains one or more horizontal line segments. Assume that, among terminal t, bend points and all vertices of obstacles in A_{ty}, exactly q points $w_1, w_2, \cdots, w_q$ are visible from line segments of $L \cap A_{ty}$ in the $(-y)$-direction, and that $y(w_1) \le y(w_2) \le \cdots \le y(w_q)$. Then the segment tree T_t has exactly q leaves, say $l_1, l_2, \cdots, l_q$ from left to right. We store point w_h

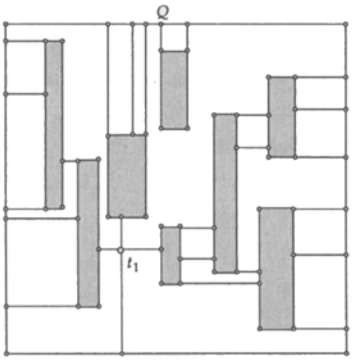

Fig. 3. Planar graph G_{t_1}.

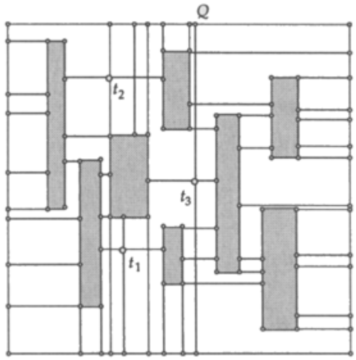

Fig. 4. Nonplanar graph G_T.

at leaf l_h for $1 \leq h \leq q$. Let $w'_1, w'_2, \cdots, w'_q$ be points on L having the same x-coordinate as $w_1, w_2, \cdots, w_q$. We store at leaf l_h, $1 \leq h \leq q$, the length of a shortest path from t to w'_h, too. We store at an internal node v of the segment tree T_t an interval $[i_1, i_2]$, where i_1 is the smallest x-coordinate of points stored at leaves which are the descendants of v, and i_2 is the largest one.

When sweep line L is set on the terminal t, we add a leaf storing t to the segment tree T_t.

When sweep line L stops on the bottom edge $be(O_i)$ of an obstacle $O_i \subseteq A_{ty}$, we add $lb(O_i)$ and $rb(O_i)$ to G_{ty} as vertices. Draw vertical line segments to $be(O_i)$ from points which are stored in T_t and whose x-coordinates are in interval $[x(lb(O_i)), x(rb(O_i))]$. Add to G_{ty} as vertices these intersection points on $be(O_i)$, and add to G_{ty} as edges these vertical line segments together with horizontal line segments on $be(O_i)$ connecting two consecutive ones among these vertices. After adding these vertices and edges to G_{ty}, we delete from T_t the leaves whose points have x-coordinates in interval $[x(lb(O_i), x(rb(O_i))]$. Furthermore we add to T_t two leaves storing $lb(O_i)$ and $rb(O_i)$.

On the other hand, when sweep line L stops on the top edge $ue(O_i)$ of an obstacle $O_i \subseteq A_{ty}$, we find a bend point $p_i^* \in ue(O_i)$ if there is. Since

$$x(p_i^*) = \frac{1}{2}\{x(ru(O_i)) + x(lu(O_i)) + d(t, ru(O_i)) - d(t, lu(O_i))\},$$

$x(p_i^*)$ can be easily computed from $d(t, ru(O_i))$ and $d(t, lu(O_i))$. Add the three vertices p_i^*, $lu(O_i)$ and $ru(O_i)$ to G_{ty}, and add the left and right edges of O_i, $[lu(O_i) - p_i^*]$, and $[p_i^* - ru(O_i)]$ to G_{ty} as edges. After adding these vertices and edges to G_{ty}, we insert to T_t three leaves storing p_i^*, $lu(O_i)$, and $ru(O_i)$.

When sweep line L stops on the upper edge of Q^∞, we execute operations similar to ones for the bottom edge of an obstacle. We thus complete the construction of G_{ty}. By Lemma 2.2, the number of the vertices in G_{ty} is $O(n)$. Since G_{ty} is a plane graph, the number of edges in G_{ty} is also $O(n)$.

Taking a union of four graphs G_{tz}, $z \in \{\pm x, \pm y\}$, we construct a plane graph G_t, where each vertex of G_{tz} is a vertex of G_t and each edge of G_{ty} is an edge of G_t. The graph G_t is also a plane graph, and hence has $O(n)$ vertices and edges. Fig. 3 illustrate G_{t_1} for the example in Fig. 1.

We similarly find the length of the shortest path connecting t and each vertex of G_t by the plane sweep.

The following lemma holds for the graph G_t.

Lemma 3.1 (a) *Let $Q \subseteq A$ be a rectangle such that the proper inside of Q does not intersect any vertical edge of G_t. Then $f^t_{L_h}$ is a straight line segment with the same slope for each horizontal line segment L_h in Q. (Hence there is no bend point of $f^t_{L_v}$ in the proper inside of Q.)*

(b) *Let $Q \subseteq A$ be a rectangle such that a proper inside of Q does not intersect any horizontal edge of G_t. Then $f^t_{L_v}$ is a straight line segment with the same slope for each vertical line segment L_v in Q. (Hence there is no bend point of $f^t_{L_v}$ in the proper inside of Q.)* □

3.2 Graph G_T

In this section, using G_t, we construct graph G_T. Furthermore we show that each vertex in $V(R(c))$ lies on an edge of G_T.

Let G_T be a union of the k graph G_{t_i} as illustrated in Fig. 4 for the example in Fig. 1. Thus the vertex set $V(G_T)$ of graph G_T satisfies $V(G_T) = \bigcup_{i=1}^{k} V(G_{t_i})$. The edge set $E(G_T)$ of graph G_T includes all edges of G_{t_i} that do not lie on $B(A \cap Q^\infty)$, that is, the boundaries of obstacles and Q^∞, and includes all line segments connecting two consecutive vertices on $B(A \cap Q^\infty)$. Each graph G_t is a plane graph, but G_T may not be a plane graph. However G_T has $O(n)$ vertices and edges, because $k = O(1)$. We assign, to each vertex p of graph G_T, the total length of the k shortest paths connecting p and all terminals, that is, $d_T(p) = \sum_{t_i \in T} d(p, t_i)$. For a vertex p of G_T, the total length $d_T(p)$ can be calculated from $d(p, t_i)$, $1 \leq i \leq k$, in $O(1)$ time. Thus, for all vertices of G_T, the assignment can be done in $O(n)$ time. For each edge e of G_{t_i}, $f^{t_i}_e$ is a straight line. However f^T_e is not always a straight line for an edge of G_T. If e is an edge of graph G_{t_i} and intersects an edge e' of graph G_{t_j} at a point $p \in \mathbb{R}^2$ in the proper inside of edges, then p is not a vertex of the graph G_T. Though $f^{t_i}_e$ is a straight line, $f^{t_j}_e$ may not be a straight line. Thus p may be a bend point of $f^{t_j}_e$, and hence p may be a bend point of f^T_e. However, if an edge e of G_T is on $B(A \cap Q^\infty)$, then there is no edge which intersects e, and hence f^T_e is a straight line l, and we assign the *slope* $s(e)$ of l to e. The other edges are assigned nothing. Note that the slope $s(e)$ can be easily calculated from $d_T(p_1)$ and $d_T(p_2)$ in time $O(1)$ for an edge $e = [p_1 - p_2]$ on $B(A \cap Q^\infty)$. Thus the assignment of the slopes $s(e)$ for edges e of G_T can be done in time $O(n)$. Hence we can do the assignment for vertices and edges of G_T in time $O(n)$. The following lemma holds for the graph G_T.

Lemma 3.2 *Every vertex $p \in V(R(c))$ lies on an edge of graph G_T.* □

3.3 Finding vertices of $R(c)$

In this section we present an algorithm to find all vertices in $V(R(c))$. Using G_T, we find all vertices in $V(R(c))$ by plane sweep in the x-direction and y-direction.

If $d_T(p) = c$ for a vertex p of a rectangular obstacle but p is not a proper vertex of $R(c)$, then p is a degenerated vertex of $R(c)$. Thus one can observe that a vertex p of an obstacle is a vertex of $R(c)$ if and only if $d_T(p) \leq c$. Therefore one can easily find all vertices of $R(c)$ that are vertices of obstacles. Furthermore, using $s(e)$, one can easily find a vertex of $R(c)$ on an edge e on the boundary of an obstacle. Thus it suffices to show how to find vertices of $R(c)$ on edges of G_T which are not on $B(A \cap Q^\infty)$.

For vertical line segments $L_1 = [p_1 - p'_1]$ and $L_2 = [p_2 - p'_2]$ in A, we say that L_1 is *properly visible* from L_2 in the $(-x)$-direction if there are points $q_1 \in (p_1 - p'_1)$ and $q_2 \in (q_2 - q'_2)$ such that $y(q_1) = y(q_2)$, $x(q_1) \leq x(q_2)$, and $[q_1 - q_2] \subseteq A$.

Let $e_h = [p_1 - p_2]$ be a horizontal edge of G_T which is not on $B(A \cap Q^\infty)$. One may assume that $x(p_1) < x(p_2)$. Let $p'_1 = (x(p_1), y(p_1) + \varepsilon)$, $p''_1 = (x(p_1), x(y_1) - \varepsilon)$, $p'_2 = (x(p_2), y(p_2) + \varepsilon)$, and $p''_2 = (x(p_2), y(p_2) - \varepsilon)$. For a small positive number $\varepsilon > 0$, (i) the rectangle $p_1 p'_1 p'_2 p_2$ is in A and does not intersect any horizontal edge of G_T except e_h, (ii) the rectangle $p_1 p_2 p''_2 p''_1$ is in A and does not intersect any horizontal edge of G_T except e_h, and (iii) there exists exactly one vertical edge of G_T on $B(A \cap Q^\infty)$ which is properly visible from a vertical line $[p'_1 - p_1]$ in the $(-x)$ direction; such an edge is denoted by $lu[e_h]$, and is called the *left upper edge of* e_h. We similarly define the *left below edge* $lb[e_h]$ *of* e_h, and define the *below left edge* $bl[e_v]$ and the *below right edge* $br[e_v]$ for a vertical edge e_v of G_T which is not on $B(A \cap Q^\infty)$.

We then have the following lemma.

Lemma 3.3 (a) *Let e_h be a horizontal edge of G_T which is not on $B(A \cap Q^\infty)$, and let p be a point on e_h, and let G_T have no vertical edge passing through p. Then $p \in V(R(c))$ if and only if $d_T(p) = c$, $s(lu[e_h]) \neq s(lb[e_h])$, and the slope of $f^T_{e_h}$ at p is not zero.*

(b) *Let e_v be a vertical edge of G_T which is not on $B(A \cap Q^\infty)$, and let p be a point on e_v, and let G_T have no horizontal edge passing through p. Then $p \in V(R(c))$ if and only if $d_T(p) = c$, $s(bl[e_v]) \neq s(br[e_v])$, and the slope of $f^T_{e_v}$ at p is not zero.* □

Using the lemma above, one can show that all vertices in $V(R(c))$ can be found by plane sweep in time $O((K + n) \log n)$. Our algorithm finds all vertices of $R(c)$ on horizontal edges in G_T by plane sweep on the x-direction, and then finds all vertices of $R(c)$ on vertical edges in G_T by plane sweep on the y-direction. (The detail is omitted in this extended abstract.)

3.4 Boundary of $R(c)$

In the previous sections we have found all vertices of $R(c)$. We find the boundary $B(R(c))$ from the vertices of $R(c)$. For the purpose, we compute the slopes of all edges $e = [w - w']$ of $B(R(c))$ incident to each vertex $w \in V(R(c))$. Then, using these slopes, we find every pair of vertices in $V(R(c))$ corresponding to the two ends of an edge of $B(R(c))$. It should be noted that the other end w' of an edge incident to w is not known. However, we can know whether an edge e of $R(c)$ is incident to w either from above or from below. (The detail is omitted in this extended abstract. The key idea is to use rotation of a coordinate system and lexicographic sorting.)

4 Conclusion

In this paper we presented an efficient algorithm for finding a feasible region $R(c)$ for given k terminals, n axis-parallel rectangular obstacles on the plane and a positive real number c. The algorithm takes $O((K+n)\log n)$ time and $O(K+n)$ space if k is a fixed constant, where K is the total number of vertices in $V(R(c))$. Since finding a single shortest path between two points on the plane with n rectangular obstacles requires time $\Omega(n \log n)$, our algorithm is quite efficient.

One can construct an algorithm to find an optimal region R^* as follows. Slightly modifying our algorithm for finding a feasible region, one can construct an algorithm to find the value $d^* = \min d_T(p)$. Finally, taking d^* as c, we can find a feasible region $R(d^*)$. Clearly $R(d^*)$ is the optimal region R^*. It is easy to observe that a connected component of an optimal region R^* is either a single point, a horizontal or vertical line segment, or an axis-parallel rectilinear polygon.

If a real number α_i is assigned to each terminal $t_i \in T$ as a weight, then the *weighted feasible region* $R_{weight}(c) = \{p \in A | \sum_{i=1}^{k} \alpha_i d(p, t_i) \leq c\}$ can be similarly found.

If the number k of terminals is not always a fixed constant, then the nonplanar graph G_T has $O(kn)$ vertices and edges. In this case one can compute in $O(k^2 n)$ time the total distance $d_T(w) = \sum_{i=1}^{k} d(w, t_i)$ for all vertices w of a graph G_T. Thus, a feasible region and an optimal region can be similarly found in time $O((K+kn)\log(kn) + k^2 n)$ using $O(K+kn)$ space.

It is rather straightforward to modify our sequential algorithm to an NC parallel algorithm which finds $R(c)$ or R^* in polylog time using a polynomial number of processors. Note that there are NC parallel algorithms for the shortest path problem [1, 6, 10] and for the plane sweep [6].

The following variations of our problem are remaining as future works:

(1) obstacles are not always rectangles but are axis-parallel polygons; and
(2) find an optimal region or a feasible region in other metric.

Acknowledgement We wish to thank Dr. H. Suzuki of Ibaraki University for fruitful discussions and helpful comments.

References

1. M. J. Atallah and D. Chen: "Parallel rectilinear shortest paths with rectangular obstacles," Comput. Geom. Theory Appl., 1, pp. 79-113, 1991.
2. K. L. Clarkson, S. Kapoor, and P. M. Vaidya: "Rectilinear shortest paths through polygonal obstacles in $O(n(\log n)^2)$ time," Proc. Third Annual ACM Symp. on Computational Geometry, Waterloo, Ontario, pp. 251-257, 1987.
3. P.J. de Rezende, D.T. Lee, and Y.F. Wu, "Rectilinear shortest paths with rectangular barriers," *Discrete and Computational Geometry,* 4, pp. 41-53, 1989.
4. E.G. Gilbert and D.W. Johnson, "Distance functions and their application to robot path planning in the presence of obstacles," *IEEE Trans. of Robotics and Automation,* RA-1, 1, pp. 21-30, 1985.
5. S. Guha and I. Suzuki: "Proximity problems for points on a rectilinear plane with rectangular obstacles," Algorithmica, 17, pp. 281-307, 1997.
6. J. JáJá, An Introduction to Parallel Algorithms, Addison Wesley, Reading, MA, 1992.
7. R. C. Larson, and V. O. Li: "Finding minimum rectilinear distance paths in the presence of barriers," Networks, 11, pp. 285-304, 1981.
8. D. T. Lee, C. D. Yang, and T. H. Chen: "Shortest rectilinear paths among weighted obstacles," Int. J. Comput. Geom. Appl., 1, pp. 109-204, 1991.
9. J. S. B. Mitchell: "L_1 shortest paths among polygonal obstacles in the plane," Algorithmica, 8, pp. 55-88, 1992.
10. P.N. Klein, "A linear processor polylog-time algorithm for shortest paths in planar graphs," *Proc. of 34th Symp. on Found. of Comput. Sci.,* pp. 259-270, 1993.
11. Y. Kusakari, H. Suzuki, and T. Nishizeki, "Finding a shortest pair of paths on the plane with obstacles and crossing areas," *Proc. of ISAAC'95, Lect. Notes in Computer Science, Springer-Verlag,* 1004, pp. 42-51, 1995.
12. Y. Kusakari, H. Suzuki, and T. Nishizeki, "An algorithm for finding a shortest pair of paths in a plane region," Trans. on Japan Soc. for Ind. and Appl. Math., Vol. 5, No. 4, pp. 381-398, 1995 (in Japanese).
13. J-C. Latombe, Robot Motion Planning, Kluwer Academic Publ., Boston, 1991.
14. T. Lengauer, Combinatorial Algorithms for Integrated Circuit Layout, John Wiley & Sons, Chichester, England, 1990.
15. T. Ohtsuki(Editor), Layout Design and Verification, North-Holland, Amsterdam, 1986.
16. F.P. Preparata, and M.I. Shamos, Computational Geometry: An Introduction, Springer-Verlag, New York, 1985.