International Journal of Computational Geometry & Applications

FINDING RECTILINEAR PATHS AMONG OBSTACLES IN A TWO-LAYER INTERCONNECTION MODEL*

D. T. Lee C. D. YANG†
Department of Electrical and Computer Engineering
Northwestern University
Evanston, IL 60208
e-mail: dtlee@ece.nwu.edu

and

C. K. WONG‡
Department of Computer Science
Chinese University of Hong Kong
Shatin, New Territories, Hong Kong
e-mail: wongck@cs.cuhk.hk

Received 25 January 1992
Revised 5 January 1996
Communicated by G. T. Toussaint

ABSTRACT

Finding the best rectilinear path with respect to the bends and the lengths of paths connecting two given points in the presence of rectilinear obstacles in a two-layer model is studied in this paper. In this model, rectilinear obstacles on each layer are specified separately, and the orientation of routing in each layer is fixed. An algorithm is presented to transform any problem instance in the two-layer model to one in a one-layer model, so that almost all algorithms for finding rectilinear paths among obstacles in the plane can be applied. The transformation algorithm runs in $O(e \log e)$ time where e is the number of edges on obstacles lying on both layers. A direct graph-theoretic approach to finding a shortest path in the two-layer model, which is easier to implement is also presented. The algorithm runs in $O(e \log^2 e)$ time.

Keywords: Rectilinear shortest path, grid graph, minimum link path, VLSI routing.

1. Introduction

When we do routing, i.e., connecting two points in the presence of obstacles, the number of bends and the length of a collision-free path have long been two important

*Supported in part by the National Science Foundation under grants CCR-8901815 and CCR-9309743.

†Current address: Avant! Corporation, 46871 Bayside Parkway, Fremont, CA 94538, e-mail: yangcd@avanticorp.com.

‡On leave from IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598.

factors for consideration. It is quite natural to seek a path of minimum total length, as it mostly reflects the timing delay or cost of the path. In some situations, we may want to minimize the number of bends in the path. For example, in motion planning when it is costly for a robot to make turns, we aim to minimize the number of bends. In VLSI routing where wires can only go in a certain direction in each layer, a bend in the path will represent a *via*, which occupies a large area. The more the vias, the less reliable it is. We therefore would like to minimize the number of bends.

Among the previous results that dealt with collision-free shortest rectilinear paths in the presence of a set of rectilinear obstacles in the plane, Clarkson, Kapoor and Vaidya[1] presented an $O(e \log^2 e)$ time and $O(e \log e)$ space algorithm, where e is the number of obstacle edges, Wu *et al.*[19] presented an $O(e \log t + t^2 \log t)$ time algorithm, where t is the number of extreme edges among those given obstacles, and Mitchell gave an $O(e \log^2 e)$ time algorithm,[13] and later improved it to $O(e \log e)$ time,[14] using a *wave-front* approach.

On the other hand, among the results that dealt with the bend factor, Ohtsuki[16] proposed an $O(e \log^2 e)$ time algorithm to find a minimum-bend path, and with the data structure of Imai and Asano,[8] the problem can be solved in $\theta(e \log e)$ time. Most recently, the authors[20] address these two cost factors in various ways. Using a graph-theoretic approach, they showed that the problems of finding the minimum-bend shortest path, the shortest minimum-bend path and the minimum cost path, where the cost is a non-decreasing function of the number of bends and the length of a path, can be solved in $O(K + e \log e)$ time, where K is $O(et)$. They also provided algorithms for finding best possible paths when one of the two factors is bounded. de Berg et al.[3] also gave an algorithm for finding a path that optimizes a linear function of both length and the number of bends, known as the *combined* metric. See Lee et al.[12] for more information on this topic.

In this paper we address the routing problem in a two-layer environment, where wires in the two layers can run in horizontal and vertical directions exclusively in each layer, and the obstacles represented as rectilinear polygons are given separately in each layer. In VLSI design, this two-layer model is widely adopted for routing. We shall find a scheme to transform any two-layer problem instance into one on a one-layer model so that most of the previously known algorithms dealing with problems in the plane are applicable, as long as the algorithms for these problems satisfy the *"alignment"* property to be defined later. Since the transformation takes only $O(e \log e)$ time, most of the algorithm complexities in the plane (single layer) follow on the two-layer model.

Alternatively we also present an algorithm for finding a shortest path in the two-layer model using a graph-theoretic approach, which runs in $O(e \log^2 e)$ time.

We organize this paper as follows. We give some preliminary definitions in Section 2. In Section 3, we introduce the idea of our problem transformation and the data structure, *skeleton*, generated by the transformation. In Section 4, we describe the details of the transformation algorithm and analyze its complexity. In Section 5, we describe an $O(e \log^2 e)$ algorithm to find a shortest path in the two-layer model without using the transformation. The conclusion is given in the last section.

2. Preliminaries

Definition 1 The *two-layer model* is defined as follows. Two layers are given, one of which allows only horizontal routing, and the other only vertical routing. Obstacles are given on each of the layers and they do not intersect on the same layer. The layer where only horizontal lines are allowed is called the *horizontal routing layer* (denoted as H_{layer}) and the one for vertical lines is called the *vertical routing layer* (denoted as V_{layer}).

Each bend of a path in this model implies a via between the two layers. Two points on different layers are called *siblings* if they have the same coordinates. We define a path in this model as follows.

Definition 2 A *path* π from the source point s to the destination point d consists of a sequence of alternating horizontal and vertical segments with *vias* connecting two *sibling* points where the segments change layer.

Obviously if we compress the two layers into one, each via on a path maps to a bend. Since no bends can appear on each layer, this is the only case when a bend is formed. In the following discussion, we only use the term *bend* to mean the junction where two sibling points meet.

3. The Problem Transformation

Our goal is to transform a problem instance in the two-layer model into one in a single layer. This transformation is mainly applied on obstacles. Intuitively an obstacle specified in one layer can be considered *transparent* while routing a line in another layer. That is, an obstacle in H_{layer} is not an obstacle for all the subpaths on V_{layer}, and symmetrically an obstacle in V_{layer} does not block any subpath on H_{layer}. We therefore aim to represent obstacles in each layer in a way that will achieve the same effect.

An obstacle in H_{layer} is equivalent to a set of infinite number of vertical segments of *zero* width covering its area. This retains the effects of the obstacle since all horizontal segments on H_{layer} will be blocked. If we copy such a representation onto V_{layer}, it does not affect anything since all vertical segments will not be blocked by these zero-width vertical segments. That is, such a representation of an obstacle on H_{layer} can be placed exactly at the same place in the vertical layer V_{layer} and it behaves exactly the same as if it is on H_{layer}. Similarly we can represent each obstacle in V_{layer} by an infinite number of *zero* width horizontal line segments covering its area. Now if we do this transformation to each obstacle and place all of those representations on a common layer, called the *goal layer* (denoted as G_{layer}), we will have an instance that is equivalent to the original one. The set of infinite number of horizontal line segments corresponding to an obstacle in V_{layer} is called the *sketch* of the obstacle. Similarly the sketch of an obstacle in H_{layer} is the set of infinite number of vertical line segments corresponding to the obstacle. The *sketch* of the set of obstacles is the union of all sketches and is denoted by $\mathcal{E}$. (Figure 1) The only problem obviously is that we cannot afford the complexity of representing an obstacle in a two-layer model by an infinite number of obstacle segments in a one-layer model. Therefore, we aim to find a finite number of segments that suffice to represent the obstacles while preserving the same effects.

Fig. 1. Sketch of obstacles in the two-layer model.

It turns out that such a finite representation can be obtained if certain *properties* of an optimal path can be enforced, as we will show below. The following lemma is obvious.

Lemma 1 *A collision-free path in a given obstacle set O in the two-layer model is also a collision-free path on the goal layer with the sketch of O representing the obstacle set.*

We now describe a *special sketch* that is finite and sufficient to represent those obstacles for algorithms that obey certain rules.

Definition 3 The *skeleton* $\subseteq \mathcal{E}$ of a set of obstacles given in the two-layer model, consisting of a set of horizontal and vertical line segments, is defined as follows: Note that a point is considered to be contained in an obstacle on H_{layer} or V_{layer}, if either the point or its sibling is contained in the interior of the obstacle.

(1) Any maximal vertical (respectively horizontal) segment contained in an obstacle on H_{layer} (respectively V_{layer}) that passes through a vertex or its sibling of an obstacle is in *skeleton*.

(2) Any maximal vertical (respectively horizontal) segment contained in an obstacle on H_{layer} (respectively V_{layer}) that passes through an endpoint of a skeleton segment is in *skeleton*.

(3) No other segment in $\mathcal{E}$ is in *skeleton*.

For example, the bold-face solid line segments in Figure 2 are the skeleton segments of the obstacles shown in Figure 1.

Fig. 2. The skeleton of the obstacles shown in Figure 1.

Definition 4 A path optimization problem is said to have the *grid* property if there exists an optimal path that lies on the grid defined by the obstacles. Such an optimal path is called a *grid* path. An algorithm for the path optimization problem with the grid property is said to have the *alignment* property if it always finds an optimal grid path consisting of segments that are *aligned* with either the source, destination, or the boundary edges of the obstacles.

The path optimization problems in the two-layer model considered in this paper all have the grid property. We shall show below that the skeleton representation of the obstacles in the two-layer model suffices for those algorithms with the alignment property to be applicable.

Toward this end we shall make an assumption, **assumption A**, that the source point or the destination point and their siblings do not lie in the *interior* of any obstacle. Suppose, for example, the destination point, d, on V_{layer} is *internal* to an obstacle, o, on H_{layer}[a] (We rule out the consideration that d also lies in the interior of an obstacle on V_{layer}, for otherwise it is *not* accessible. In fact if a point and its sibling are both in the interior of an obstacle in either layer, then this point is *not* accessible from outside.) It is obvious that only vertical segments from outside of o can reach d. Therefore, we can simply replace d by two points d_1 and d_2 which

[a] More precisely it is the sibling of d that is internal to an obstacle on H_{layer}. We will omit the word, *sibling*, when it is understood.

are the intersection points between the boundary of o and the vertical line passing through d. If either one is in the interior of an obstacle, then it is ignored. (see Figure 1, where d_2 is ignored.) Do the same thing to the source point s if it is also *internal* to some obstacle; let the two possible source points be s_1 and s_2. The problem is then reduced to finding the best solution from the solutions to at most four subproblems, i.e., paths from X to Y where X $\in \{s_1, s_2\}$ and Y $\in \{d_1, d_2\}$.

With **Assumption A**, we are able to prove the following important lemma.

Lemma 2 *An optimal (with respect to bends and length) path produced by any one-layer algorithm with the alignment property for problem P_{new}, which includes the source and the destination points and the skeleton representation of the obstacles given for problem P_{old} in the two-layer model, is also a collision-free path for P_{old}, if P_{old} satisfies Assumption A.*

Proof. From Lemma 1, we know that any feasible path in P_{old} is feasible in P_{new}. It is sufficient to prove that any optimal path with respect to the bends and the length found by solving the P_{new}, is also a feasible path in P_{old}. We prove it by contradiction. Assume we have a path, π, optimal in terms of bends and/or length found in P_{new}, but it is not a feasible path in P_{old}. That is, π has collision with some obstacle in any of the two layers, which means there exists a vertical or horizontal segment of π contained in an obstacle in V_{layer} or H_{layer} respectively (Refer to Figure 3).

Fig. 3. Illustration for the proof of Lemma 1.

Without loss of generality, let us just consider the former case. Let the vertical segment of π intersecting an obstacle in V_{layer} be $v(a, b)$, where a and b are the two end points of v. First, v cannot partially be contained in an obstacle (denoted o), since the horizontal segments of the boundary of o are in the skeleton and thus cannot be intersected by π. Now, since v is totally contained in o, it must make turns on points a and b inside o. If the algorithm generating π has the alignment property, there must be a vertical segment, $v'(a', b')$, in the skeleton aligned with

v. The two endpoints a' and b' may or may not be inside of o. If a' is inside of o, then by the definition of skeleton, there must be another horizontal skeleton segment passing through a'; otherwise a' must be at the intersection of $v'(a', b')$ and a horizontal segment, $\overline{A, B}$, along the boundary of o (Figure 3). The same is true for b'. The path π must have entered the sketch of o below segment $\overline{A, B}$ at some point m, and exit at some point n, since the source and destination of π must not be inside of any obstacle, according to **Assumption A**. A U-shaped subpath on π must exist in the subpath from m to n, where m and n are defined as above. This violates that π is minimum in terms of bends and length (We can always shrink the U-shaped subpath as in Ref. [20] to form a better path). The lemma is thus proved. □

From the above lemmas, one can see that any shortest path found by an algorithm with the alignment property in a one-layer model must also be a shortest path in the original two-layer model. We have the following theorem.

Theorem 1 *The X-path found by an algorithm with the alignment property on a one-layer problem instance obtained from the two-layer model problem instance in which the set of obstacles is represented by its skeleton, must also be an X-path in the original problem, where $X \in \{$ shortest, minimum-bend, minimum (y-distance) shortest, shortest (y-distance) minimum-bend, minimum-cost, bounded bends (y-distance) shortest, bounded (y-distance) length minimum-bend $\}$.*

We now show that the number of line segments in the *skeleton* is *linear* in e and present an algorithm to perform the transformation. Let us first give a characterization of the line segments in the *skeleton*.

Definition 5 Let the line segments contained in the *skeleton* be classified as follows:

Type 1 segments are those that contain some vertices (or their siblings) of obstacles,

Type i segments are those that contain an endpoint of a **type (i-1)** segment, for $i > 1$.

If a segment is of **type** i and **type** j, $i < j$, it is classified as **type** i.

Lemma 3 *There are only three types of segments in the skeleten of a given set of obstacles O in the two-layer model and the total number of segments is $O(e)$, where e is the number of edges in O.*

Proof. By the definition of the skeleton segments, a **type** i segment is generated by an endpoint of a **type** $i-1$ segment, for $i = 1, 2, 3$, where an obstacle vertex (or its sibling) is considered to be an endpoint of a **type 0** segment.

We first prove that there is no skeleton segment of **type 4**. Let m be a horizontal **type 2** segment (the vertical case is treated symmetrically) contained in an obstacle o on V_{layer} and passing through an endpoint t_1 of a vertical **type 1** segment t contained in obstacle u on H_{layer} according to the definition (refer to Figure 4(a)). The boundary segment of obstacle u that lies in the interior of o and contains m cannot contain a vertex, for otherwise m will be classified as **type 1**. Therefore, the endpoints of m, denoted m_1 and m_2 must be on the boundary of both u and o. Suppose that there is a vertical **type 3** segment g generated from m_1 and contained in u. Since g, which is of **type 3**, cannot contain any obstacle vertex, it must be on a vertical boundary edge of o. Let the other endpoint of g be g_1 and let f be a presumed **type 4** segment generated by g_1. f must be a horizontal maximal segment contained in o and pass through g_1. Again, since f is of **type 4**, the boundary segment of u which contains f cannot make any turn in the interior of obstacle o. We have two possibilities.

(a) (b)

Fig. 4. Illustration for the proof of Lemma 3.

Referring to Figure 4 the other endpoint, f_1, of f has abscissa either (1) no greater than or (2) greater than that of m_2. In case (1), the boundary edge of o passing through f_1 must be vertical and make a turn somewhere below m since m is totally contained in o. Once the boundary makes a turn, there is a vertex lying inside u, which means there is a **type 1** vertical segment covered by u which has f_1 as its endpoint. Therefore, f should be of **type 2** instead of **type 4**, a contradiction. In case (2), i.e., f goes farther than m_2, as shown in Figure 4(b). Imagine dragging segment g to the right. We must either hit an obstacle vertex or reach t, since t is of **type 1**. Assume that we hit vertex a. There must be a vertical **type 1** segment passing through a which has an endpoint on f. This would make segment f a **type 2** segment, a contradiction. If t is encountered, then the other endpoint t_2 of t must lie on f. Since t is of **type 1**, f is of **type 2**, a contradiction. Since there are only three types of skeleton segments and the number of **type** i segments is linearly proportional to the number of **type** $i-1$ segments, the total number of segments in the skeleton is $O(e)$. The lemma is proved. $\square$

4. The Algorithm

We now describe the algorithm **ToSkeleton** which transforms a set of given obstacles on H_{layer} and V_{layer} into the corresponding *skeleton* representation on G_{layer}.

Definition 6 Define a *horizontal* (*resp. vertical*) *projection* of a point p in a set of obstacles O in either layer to be a point q on the boundary of an obstacle in O such that the horizontal (resp. vertical) segment $\overline{p,q}$ is maximal and does not cross any boundary edge of any obstacle in O.

Definition 7 Define the *internal projection,* x_{iProj}, of a point x internal to an obstacle o on H_{layer} (respectively V_{layer}) to be the set containing at most two vertical (respectively horizontal) projections on the boundary of the obstacle o to which x is *internal.*

[**Algorithm ToSkeleton**]

1. Preprocess separately the set of segments which form the boundary of the obstacles in each layer so that for any given point q one can find its horizontal or vertical projections of q in each layer using the planar-subdivision structure for point location.[5,6]

2. Let $\mathcal{K}$ be the skeleton and $\mathcal{Q}_1^h$ and $\mathcal{Q}_1^v$ be queues storing the endpoints of skeleton segments.
For each concave vertex[b] p of an obstacle o_h in H_{layer} find its vertical projection p_1 such that the line segment $\overline{p,p_1}$ is maximal and lies totally in o_h. Include $\overline{p,p_1}$ as a **type 1** segment in $\mathcal{K}$ and insert p_1 into $\mathcal{Q}_1^v$, if it does not coincide with a vertex of o_h. If p^s, the sibling of vertex p of an obstacle o_h in H_{layer}, lies in the interior of an obstacle o_v in V_{layer}, find its projection $p^s_{iProj} = \{p^s_1, p^s_2\}$ and insert both points in p^s_{iProj} into $\mathcal{Q}_1^h$ and include $\overline{p^s_1, p^s_2}$ as a **type 1** segment in $\mathcal{K}$. Analogously for each vertex p' of o'_v in V_{layer}, find its horizontal projection p'_1, such that the line segment $\overline{p', p'_1}$ is maximal and lies totally in o'_v. Include $\overline{p', p'_1}$ as a **type 1** segment in $\mathcal{K}$ and insert p'_1 into $\mathcal{Q}_1^h$, if it does not coincide with a vertex of o'_v. If p'^s, the sibling of vertex p' of an obstacle o_v in V_{layer}, lies in the interior of an obstacle o_h in H_{layer}, find its projection $p'^s_{iProj} = \{p'^s_1, p'^s_2\}$ and insert both points in p'^s_{iProj} into $\mathcal{Q}_1^v$ and include $\overline{p'^s_1, p'^s_2}$ as a **type 1** segment in $\mathcal{K}$.

3. For each point q in $\mathcal{Q}_1^v$, determine if its sibling q^s lies in the interior of an obstacle o_v in V_{layer}. If so, compute its projection $q^s_{iProj} = \{q_1, q_2\}$ in o_v; include segment $\overline{q_1, q_2}$ in $\mathcal{K}$ as a **type 2** segment and insert q_1, and q_2 in $\mathcal{Q}_2^h$ if it is not a vertex of o_v. If q^s lies outside of all obstacles in V_{layer}, ignore it. Similarly, for q' in $\mathcal{Q}_1^h$ determine if its sibling q'^s lies in the interior of an obstacle o_h in H_{layer}. If so, compute its projection $q'^s_{iProj} = \{q'_1, q'_2\}$ in o_h; include segment $\overline{q'_1, q'_2}$ in $\mathcal{K}$ as a **type 2** segment and insert q'_1, and q'_2 into $\mathcal{Q}_2^v$

[b] A vertex of an obstacle is said to be *concave*, if the interior angle is equal to 270°.

if it is not a vertex of o_h. If q'^s lies outside of all obstacles in H_{layer}, ignore it.

4. Do the same as in Step 3 for points in $\mathcal{Q}_2^v$ and $\mathcal{Q}_2^h$ to generate **type 3** segments.

Lemma 4 *Algorithm ToSkeleton runs in $O(e \log e)$ time where e is the total number of edges of the obstacles.*

Proof. We need $O(e \log e)$ time to preprocess the edges of the obstacles in each layer so that for any point one can compute its horizontal or vertical projections in $O(\log e)$ time.[5,6] Steps 2, 3 and 4 each takes $O(e \log e)$ time since there are only $O(e)$ vertices and $O(e)$ endpoints in $\mathcal{Q}_i$, for $i = 1, 2$. □

We have now obtained an equivalent representation of the obstacles in two layers by skeleton with $O(e)$ segments in one layer. We now present the algorithm for finding an optimal path as follows.

[**Algorithm 2To1**] Given the source s, the destination d and the obstacle set O on two layers. We perform the following to find the goal path.

1. Find the *internal projections*, s_{iProj}, of s if it is internal to any obstacle and if any element in s_{iProj} is internal to an obstacle, remove it from the set. If s_{iProj} is empty, then s is not accessible. Let $s_{iProj} = \{s\}$ if s is external to any obstacle in any layer. d_{iProj} is similarly defined. The distances from points in s_{iProj} to s and from points in d_{iProj} to d are computed and stored.

2. Apply algorithm **ToSkeleton** to find the skeleton of O.

3. Find the external contour[c] of the skeleton, using an algorithm described in Ref. [17] in $O(e \log e)$ time. Figure 5 is the external contour of the segments in Figure 2[d].

4. Find an optimal path from a point in s_{iProj} to a point in d_{iProj} using the best known one-layer algorithm with the alignment property. Output the best path from s to d taking into consideration the distance and bends from s to its internal projections, if any, and from d to its internal projections, it any. All horizontal segments of the path are on the H_{layer} and all vertical segments are on the V_{layer}.

Note that after finding the external contour in Step 3, we obtain a set of non-intersecting obstacles with $O(e)$ edges as shown in Ref. [17]. Since the algorithms reported in Refs. [1,2,13,14,15,16,20,21,22] have the alignment property, we have the following results.

Theorem 2 *In the two-layer model where the routing orientation is fixed in each layer, the following problems can be solved:*
(1) Finding a shortest path can be done in $O(e \log^{3/2} e)$ time[2,11,18] *or in $O(e \log e)$*

[c]The external contour of a set of line segments $\mathcal{K}$ is the boundary between $\mathcal{K}$ and the unbounded region of the plane.
[d]Note that we may represent each of the skeleton segments by a very *thin* rectangle so that the external contour algorithm described in § 8.7 of Ref. [17] can be applied.

Fig. 5. The external contour of the skeleton of Figure 2.

time.[14]

(2) Finding a minimum-bend path can be done in $\theta(e \log e)$ *time.*[16]

(3) The shortest minimum-bend path, the minimum-bend shortest path and the minimum cost path in Ref. [22], can be found in $O(e \log^{3/2} e)$ *time and space.*

(4) A shortest y*-distance path can be found in* $\theta(e \log e)$ *time.*[15]

(5) The shortest minimum-bend path, the minimum-bend shortest y*-distance path and the shortest* y*-distance minimum-bend path can be found in optimal* $\theta(e \log e)$ *time.*[21]

Other results presented in Refs. [3,4,9] are also applicable for solving problems in the two-layer model.

5. Direct Method for Finding a Shortest Path

The results in the previous section are based on the transformation from the two-layer model problem into a one-layer model problem. A disadvantage of such a transformation is the overhead involved as well as possible loss of information in the original instance. For example, in Wu, *et al.*[19] and Yang, *et al.*,[20,22] the number of the extreme edges in the contour of the skeleton is an important parameter for the run time. However, after the transformation the number of extreme edges is at least the same as the the number of obstacles. The final complexity does not depend on the number of extreme edges of the given obstacles any more. For some problems, a more efficient and direct approach can be taken so that the overhead of transformation can be eliminated. In the following, we present an algorithm which computes a shortest-path-preserving graph from the input in the two-layer model. Though the time complexity is $O(e \log^2 e)$, the algorithm is simpler to implement and faster when the input size is considerably small.

Let V be the set of obstacle vertices. The following is our starting graph which preserves at least a shortest path. The reader is referred to Refs. [2,11,18] for a basic graph structure for similar problems on one layer.

Definition 8 Define the graph $G0 = (V_{G0}, E_{G0})$ as follows:

1. s, d and all $X \in V$ are in V_{G0}.

2. All projection points in X_{iProj}, for all $X \in \{s, d\} \bigcup V$, are in V_{G0}.

3. Let R_{pq} denote the closed rectangular area with $\overline{pq}$ as its diagonal. Add edge (p, q) for all $p, q \in V_{G0}$ into E_{G0} if and only if:

  3.1. no vertex in V_{G0}, except p and q, is in R_{pq}, and

  3.2. no obstacle in V_{layer} crosses the left and right boundaries of $R_{p,q}$ and

  3.3. no obstacle in H_{layer} crosses the top and bottom boundaries of R_{pq}.

In other words, we connect two vertices, p and q, with an edge in $G0$ if there is no other vertex in R_{pq} and there exists a staircase path, π, connecting p and q with length equal to the rectilinear distance between p and q. Figure 6 gives the graph $G0$ for the input shown in Figure 1 and Figure 10 shows an example of R_{pq}.

Fig. 6. Shortest-path-preserving graph $G0$ of the obstacles in Figure 1.

Lemma 5 *The graph $G0$ is a shortest-path-preserving graph in the two-layer model. That is, there exists at least one rectilinear shortest path from s to d embedded in graph $G0$.*

Proof. Assume π is a shortest path from s to d in the two-layer model. Let a U-shaped subpath of π be a set of three consecutive segments, s_1, s_2 and s_3 with s_1 and s_3 lying on the same side of the line containing s_2 (Figureřeffig:6). Without loss of generality, for each U-shaped subpath, $s_i, i = 1, 2, 3$, we may assume that s_2

must coincide with an edge e of an obstacle o, for otherwise we can shorten the path by moving it toward the side containing s_1 and s_3 (Figure 7). Furthermore, when s_2 coincides with e, if there is no vertex or projection point on e (Figure 7), then s_1 must cross an obstacle strip o, and s_2 must go along an edge of o and s_3 cross o again. We can directly connect s_1 and s_3 before they cross o and obtain a shorter path. This indicates that there must be a vertex or a projection point $v \in V_{G0}$ on s_2 for each U-shaped subpath and the U-shaped subpath can be broken into two at v.

Fig. 7. Segment s_2 must coincide with an edge of an obstacle.

In doing so, π can be considered a path consisting of a sequence of subpaths without U-shapes between vertices in V_{G0}. In Figure 8(a) for example, $\pi_{sm'}$ and $\pi_{m'd}$ are two U-shape-free subpaths of π_{sd}. Now consider a U-shape-free subpath from p to q, with $q.y > p.y, q.x > p.x$ without loss of generality. We shall show that each such path can be realized by an equivalent path embedded in graph $G0$. Without increasing the length, we can drag every horizontal segment downward/upward (and respectively drag every vertical segment rightwards/leftwards) so as to make it aligned with an edge e of some obstacle on H_{layer} (resp. on V_{layer}) (Figure 8(b)). If e contains a vertex $r \in V_{G0}$ in the overlapped portion with the path segment, then we may restrict ourselves to the subpath from p to r and repeat the following arguments. Thus e must be a horizontal edge of an obstacle on H_{layer} (resp. a vertical edge of an obstacle on V_{layer}) containing no projection points. This means that we can drag the segment further past the obstacle vertically (resp. horizontally) without increasing the path length. Therefore, if there is any vertex in V_{G0} that lies in R_{pq}, we can always drag the path between p and q to touch such a vertex (e.g., vertex r' in Figure 8(b)). Consequently, we can find a final path, π^*, that can be decomposed into subpaths between vertices, p' and q', of V_{G0} such that each subpath is either a vertical segment, a horizontal segment or an L-shaped segment, i.e., a vertical segment followed by a horizontal segment or vice versa, and $R_{p'q'}$ contains no other vertices in V_{G0}. It is obvious that all such subpaths are in $G0$.

Thus $G0$ preserves the shortest path. □

(a) (b)

Fig. 8. Illustration for the proof of Lemma 5.

The graph $G0$ has $O(e)$ vertices and $O(e^2)$ edges, which yields an $O(e^2)$ algorithm[7] for finding a shortest path in $G0$. We now introduce the so-called Steiner points to reduce its size and the time complexity of the algorithm. As in Refs. [2,11,18], we generate Steiner points in the graph in a recursive manner. We intend to reduce the number of edges at the expense of the number of vertices in $G0$ while supporting all the connections as in $G0$. There are two kinds of Steiner points: in addition to the ones generated on the cut lines as in Refs. [2,11,18] we have the *boundary Steiner point*, that are on obstacle boundaries. We will define these Steiner points in the algorithm described later.

Definition 9 Two points are said to be *horizontally* (*vertically*) *visible* to each other if there is a horizontal (vertical) line connecting them without being blocked by any obstacle in H_{layer} (V_{layer}). Two points are *visible* from each other if they are horizontally or vertically visible from each other. A point p is said to be visible from a vertical (horizontal) line L if there exists a point p' on L such that p and p' are visible from each other.

It is clear that every point p *inside* an obstacle has at most two points in p_{iProj}. We now define the path-preserving graph $G1$ to be the graph generated by the following algorithm:

Algorithm generateG1:

(Output graph $G1 = (V_{G1}, E_{G1})$)

1. Let V' be $\{s, d\} \bigcup V$. Find the set of internal projections X_{iProj}, for all concave vertices $X \in V'$. Let $\mathcal{U}$ be the union of V' and these projection points. Add $\mathcal{U}$ to V_{G1}.

2. For all concave vertices x add edge $\overline{x, x'}$ into E_{G1}, where x' is the projection of x; delete x from $\mathcal{U}$.

3. Draw a vertical cut line $\mathcal{L}$ with abscissa equal to the median of all the points in $\mathcal{U}$.

3.1 (See Figure 9) For a point x in $\mathcal{U}$, if it is visible from $\mathcal{L}$ with projection point x' on $\mathcal{L}$, and if x' is not *internal* to an obstacle in V_{layer}, add x' into V_{G1} as a Steiner point and add (x, x') to E_{G1}. If x' is *internal* to some obstacle in V_{layer}, let $x'_{iProj} = \{x'', x'''\}$ add x'', x''' into V_{G1} as two *boundary Steiner* points and add $\overline{x'', x'''}$ and $\overline{x, x^*}$ to E_{G1} where x^* is the nearer one to x of x'' and x'''. (In Figure 10, for example, we add $\overline{p, p''}$, $\overline{p'', p'''}$, $\overline{q'', q'''}$ and $\overline{q''', q}$ into E_{G1}.) If x' is on the boundary of some obstacle in V_{layer}, then add x'' to V_{G1} as a *boundary Steiner* point and add $\overline{x, x^*}$ to E_{G1}, where x^* is the nearer one to x of x' and x''.

3.2 Insert edges to E_{G1} for any two consecutive visible Steiner points on $\mathcal{L}$. (In Figure 9, $\overline{x', y'}$ is added into E_{G1}, whereas in Figure 10, both p' and q' are internal to the same obstacle in V_{layer}, i.e., they are not visible to each other, so there is no edge between p' and q' in E_{G1}.)

Fig. 9. Introducing Steiner point x' and edge $\overline{x, x'}$.

4. Do step 3.1 and step 3.2 recursively to the points of $\mathcal{U}$ on the right side of $\mathcal{L}$ and then to those on the left side of $\mathcal{L}$.

5. Traverse the boundaries of all the obstacles, and if any two consecutive boundary Steiner points or points in $\mathcal{U}$ are visible to each other, then insert an edge between them into E_{G1}. (In Figure 10, $\overline{p'', q''}$ and $\overline{p''', q'''}$ are added.)

Fig. 10. Introducing boundary Steiner points p'', p''', q'' and q''' and edges $\overline{p, p''}$ and $\overline{q, q'''}$.

Lemma 6 *Graph $G1$ has $O(e \log e)$ vertices and edges.*

Proof. Besides the original $O(e)$ vertices in V, others in V_{G1} are the projections, the Steiner points, and the boundary Steiner points. The number of projections is $O(e)$ since one vertex x can generate at most two projections in x_{iProj}. At each recursive level, only $O(e)$ Steiner points and boundary Steiner points are created, so a total of $O(e \log e)$ Steiner points is generated. The edges are created for connecting those Steiner points on the vertical cut lines and connecting adjacent points in $\mathcal{U}$ or the boundary Steiner points, so the number of edges is of the same order of the number of Steiner points. □

Lemma 7 *Graph $G1$ supports all the connections provided by edges in $G0$.*

Proof. (We refer the reader to the proof of a similar algorithm in Refs. [2,11]) Refer to Figure 10. For any two vertices, p and q, which are connected by an edge $\overline{p, q}$ in $G0$, there will be a cut line $\mathcal{L}$ separating them in some recursive step. In the situation described in the definition of $G0$, p and q with projections p' and q' on $\mathcal{L}$ respectively are visible from $\mathcal{L}$ We shall now show that in $G1$ there is a *path* connecting p and q, thus supporting the edge $\overline{p, q}$ in $G0$. If p' and q' are visible to each other, then the edges $\overline{p, p'}$, $\overline{p', q'}$, and $\overline{q', q}$ will provide the connection. The problem is that p' and q' may be *internal* to some obstacle, o on V_{layer}, and thus there is no edge connecting them. However, according to the algorithm, we have created boundary Steiner points, p'', p''' for p' and q'', q''' for q', which are on the boundary of o, and we also have created edges connecting adjacent boundary Steiner points. Therefore a path connecting p and q, i.e., $\overline{p, p''}, \overline{p'', q''}, \overline{q'', q'''}$ and $\overline{q''', q}$, is thus provided. □

Lemma 8 *The algorithm generate$G1$ runs in $O(e \log^2 e)$ time.*

Proof. Most of the steps are the same as those in Refs. [2,11,18] which have $O(e \log e)$ time complexity. The only differences are to query if the projections on the cut lines lie in the interior of obstacles, to compute the boundary Steiner points and to provide connections between these points along the boundaries. One can preprocess those obstacles in two different layers respectively, in $O(e \log e)$ time, so the projection queries can be done on different layers and the computation of the boundary Steiner points can be done in $O(e \log^2 e)$ time. Each of the $O(e \log e)$ boundary Steiner points will be inserted into a sorted list associated with the edge on which these boundary Steiner points lie. Each insertion takes $O(\log e)$ time, and thus $O(e \log^2 e)$ time suffices to insert all boundary Steiner points. Finally, to create edges for adjacent points in $\mathcal{L}$ and the boundary Steiner points, we need to traverse all lists associated with edges, which takes only $O(e \log e)$ time. Therefore, the overall time complexity is $O(e \log^2 e)$. □

We therefore conclude with the following theorem.

Theorem 3 *The shortest rectilinear path problem in two-layer model can be computed in $O(e \log^2 e)$ time using $O(e \log e)$ space, where e is the number of edges on obstacles, without using the* **2To1** *transformation.*

6. Conclusion

In this paper we have introduced a general transformation which transforms a problem instance in the two-layer model into one on a plane in $O(e \log e)$ time, where e is the size of the input, so that most of the previous one-layer path-finding algorithms can be applied. We have also presented an $O(e \log^2 e)$ time algorithm using a graph-theoretic approach, which solves the shortest path problem in the two-layer model without using the transformation. Whether or not a more efficient, direct approach to the shortest path and other path-finding problems in the two-layer model can be obtained remain to be seen.

References

1. K. L. Clarkson, S. Kapoor and P. M. Vaidya, "Rectilinear shortest paths through polygonal obstacle in $O(n \log^2 n)$ time" *Proc. 3rd ACM Symp. on Computational Geometry*, 1987, pp. 251-257.
2. K. L. Clarkson, S. Kapoor, and P. M. Vaidya, "Rectilinear shortest paths through polygonal obstacles in $O(n \log^{3/2} n)$ time," unpublished manuscript.
3. M. de Berg, M. van Kreveld, B. J. Nilsson, and M. H. Overmars, "Shortest Path Queries in Rectilinear Worlds," *Int'l J. Computational Geometry & Applications*, 3,2 Sept. 1992, pp. 287-309.
4. P. J. deRezende, D. T. Lee, and Y. F. Wu, "Rectilinear shortest paths with rectangular barriers," *Discrete and Computational Geometry*, 4, 1989, pp. 41-53.
5. H. Edelsbrunner, L. J. Guibas, and J. Stolfi, "Optimal Point Location in a Monotone Subdivision," *SIAM J. Computing,* 15,2 May 1986, pp. 317-340.

6. D. Kirkpatrick, "Optimal Search in Planar Subdivisions" *SIAM J. Computing*, 12,1, (Feb. 1983), pp. 28-35.
7. M. L. Fredman and R. E. Tarjan, "Fibonacci Heaps and Their Uses in Improved Network Optimization Problems," *J. ACM*, 34 1987, pp. 596-615.
8. H. Imai and T. Asano, "Dynamic Segment Intersection Search with Applications," *Proc. 25th IEEE Symp. on Foundations of Computer Science*, Singer Island, Florida, pp. 393-402, 1984.
9. R. C. Larson and V. O. Li, "Finding Minimum Rectilinear Distance Paths in the Presence of Barriers," *Networks*, 11,1981, pp. 285-304.
10. D. T. Lee, "Proximity and Reachability in the Plane," PhD. Dissertation, University of Illinois, 1978.
11. D. T. Lee, C. D. Yang and T. H. Chen, "Shortest Rectilinear Paths among Weighted Obstacles," *Int'l J. Computational Geometry & Applications*, 1,2, 1991, pp. 109-124.
12. D. T. Lee, C. D. Yang and C. K. Wong, "Rectilinear Paths among Rectilinear Obstacles," *Discrete Applied Math.*, 70 (1996), 185-215.
13. J. S. B. Mitchell, "L_1 Shortest Paths among Polygon Obstacles in the Plane," *Algorithmica*, 8,1, 1992, pp. 55-88.
14. J. S. B. Mitchell, "Shortest Paths among Obstacles in the Plane," *Proc. 9th ACM Symp. on Comput. Geometry,* May 1993, pp. 308-317.
15. T. Ohtsuki and M. Sato, "Gridless Routers for two layer Interconnection," *IEEE internat. Conf. on Computer Aided Design*, pp. 76-78, Nov. 1984.
16. T. Ohtsuki, "Gridless Routers – New Wire Routing Algorithm Based on Computational Geometry," *International Conference on Circuits and Systems*, China, 1985.
17. F. P. Preparata and M. I. Shamos, Computational Geometry, *Springer-Verlag*, NY, 1985.
18. P. Widmayer, "Network design issues in VLSI," manuscript, 1989.
19. Y. F. Wu, P. Widmayer, M. D. F. Schlag, and C. K. Wong, "Rectilinear Shortest Paths and Minimum Spanning Trees in the Presence of Rectilinear Obstacles," *IEEE Trans. Comput.*, 1987, pp. 321-331.
20. C. D. Yang, D. T. Lee and C. K. Wong, "On Bends and Lengths of Rectilinear Paths: A Graph-Theoretic Approach," *Int'l J. of Computational Geometry & Applications*, 2,1 (March 1992), pp. 61-74.
21. C. D. Yang, D. T. Lee and C. K. Wong, "On Bends and Distances of Paths among Obstacles in Two-Layer Interconnection Model," *IEEE Trans. Comput.*, 43,6 June 1994, pp. 711-724.
22. C. D. Yang, D. T. Lee and C. K. Wong, "Rectilinear Path Problems among Rectilinear Obstacles Revisited," *SIAM J. Computing*, 24,3, (June 1995), pp. 457-472.