An Efficient Direct Approach for Computing Shortest Rectilinear Paths among Obstacles in a Two-Layer Interconnection Model

Danny Z. Chen* Jinhui Xu*

Abstract

In this paper, we present a direct approach for routing a shortest rectilinear path between two points among a set of rectilinear obstacles in a two-layer interconnection model. The previously best known direct approach for this problem takes $O(n \log^2 n)$ time and $O(n \log n)$ space. By using integer data structures and an implicit graph representation scheme, we reduce the time bound to $O(n \log^{3/2} n)$ while maintaining the space bound to $O(n \log n)$. Comparing with the indirect approach for the problem, our algorithm is simpler to implement and faster when the input size is moderate.

1 Overview

The problem of computing shortest rectilinear paths among a set of obstacles is an important problem in computational geometry and often arises in applied fields such as robot motion planning and VLSI wire routing. Considerable work has been done on solving this problem (see the survey [4] for results on this problem). In this paper, we consider one of its variations in VLSI: Routing a shortest rectilinear path between two points, s and d, in a widely adopted two-layer interconnection model. This model consists of two layers: The horizontal routing layer H_{layer} (allowing only horizontal routing) and the vertical routing layer V_{layer} (allowing only vertical routing). On each layer, there are disjoint rectilinear non-penetrable obstacles. Each bend of a rectilinear path in this model implies a *via* between the two layers.

Lee, Yang and Wong [5] recently proposed two interesting approaches for this problem. The first approach (which we call the *indirect approach*) is to transform, in $O(n \log n)$ time and space, a problem instance of n obstacle edges in the two-layer model into one in a single layer in which both horizontal and vertical routings are allowed. Then by using any one-layer shortest path algorithm (e.g., [2, 6, 8]), a shortest rectilinear path in the two-layer model can be obtained in either $O(n \log^{3/2} n)$ or $O(n \log n)$ time. However, as pointed out in [5], the transformation may incur heavy overhead and cause possible loss of information in the original problem instance (e.g., the number of extreme edges). Thus, the second approach (which we call the *direct approach*) was presented in [5], giving an easier-to-implement and faster algorithm for moderate input size. The direct approach first constructs a shortest-path-preserving graph G_1, and then uses Fredman and Tarjan's shortest path algorithm to compute a shortest path in G_1. This approach takes $O(n \log^2 n)$ time and $O(n \log n)$ space [5].

The construction of G_1 uses two types of *Steiner points*, called *cut-line Steiner points* and *boundary Steiner points*. G_1 is built in a way similar to those in [2, 3]. The main difference is that, for example, [5] needs to query whether each obstacle vertex's projection point on a vertical cut-line is inside an obstacle on the vertical routing layer V_{layer}. If it is, then [5] uses two boundary Steiner points (on the boundary of that obstacle) instead of one cut-line Steiner point on the cut-line, and connects these boundary Steiner points with their neighbors along that obstacle's boundary. The implementation of these queries in [5], generating boundary Steiner points and sorting them along the corresponding obstacle boundaries, takes $O(n \log^2 n)$ time, and this is the dominating time cost of the direct approach of [5]. One of our ideas is that, by using the integer data structures [7] and plane sweeping, this dominating query part can be carried out in $O(n \log n \log \log n)$ time. Thus G_1 can also be constructed in this time bound.

Once G_1 is obtained, a technique in [2, 3] for reducing the number of cut-line Steiner points (and hence the vertices of G_1) can be applied. But, a disadvantage of this technique is that it increases the number of edges of G_1 by a factor of $\log^{1/2} n$, and hence increases the space bound from $O(n \log n)$ to $O(n \log^{3/2} n)$. We show that the number of boundary Steiner points can also be reduced. Further, by applying the scheme of distance tables [1] to both types of Steiner points, we are able to remove the $\log^{1/2} n$ factor from the space bound. Therefore, the total time of our algorithm is $O(n \log^{3/2} n)$ and the total space is $O(n \log n)$.

2 Constructing the Graph G_1

In this section, we show our ideas for generating the two type of Steiner points and sorting boundary Steiner points along their corresponding obstacle boundaries in $O(n \log n \log \log n)$ time, Other details of our algorithm are left to the full paper. We assume all obstacles are rectangles, since the rectilinear obstacles can be partitioned into $O(n)$ rectangular obstacles [5].

The Steiner points are generated by projecting the set of obstacle vertices and s, d (the union of them is denoted by U) to $O(n)$ (say) vertical cut-lines, with each vertex projecting to at most $O(\log n)$ cut-lines. A point $p \in U$ is projected to one of the $O(\log n)$ cut-lines, say L, only when p is horizontally visible from L. If it is, then we need to check whether the projection point, p', of p is internal to any obstacle on V_{layer}. If

*Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, USA. {chen,jxu}@cse.nd.edu. This research was supported in part by the National Science Foundation under grant CCR-9623585.

not, then p' becomes one of the cut-line Steiner point of L. Otherwise (p' is inside an obstacle, say o, on V_{layer}), at most two boundary Steiner points, p'' and p''', are generated on the left and right boundaries of o. To determine whether p is horizontally visible from L, we associate each vertex p in U with its two horizontal projection points, p_{hpro1} and p_{hpro2}, in the set of rectangular obstacles on the horizontal routing layer H_{layer}. Thus the visibility of p to L can be checked by comparing the x-coordinates of p_{hpro1}, p_{hpro2}, and L.

In order to process the queries on whether p is inside an obstacle on V_{layer} and to generate boundary Steiner points efficiently, we use the integer data structure [7] and plane sweeping. In [7], van Emde Boas showed that when the elements of a priority queue are from a universe $\{1, 2, \ldots, u\}$, there is a data structure which supports the following operations with $O(\log \log u)$ time per operation: **Insert**, **Delete**, **Member**, **Empty**, **Min**, **Max**, **Predecessor**, and **Successor**. To apply this data structure to our problem, we first sort all vertices in U by their x-coordinates and y-coordinates respectively, and use their ranks in the two sorted lists as the new coordinates of the vertices. Then, we sweep a horizontal line from bottom to top to generate the two types of Steiner points. The sweeping algorithm maintains two integer priority queues, Q_v and Q_h, to store the obstacle edges intersected by the sweep line on V_{layer} and H_{layer}, respectively. Each obstacle edge in the two queues is represented by its x-coordinate (i.e., an integer). When a point $p \in U$ is met by the sweep line, the sweeping algorithm projects p to all of its $O(\log n)$ vertical cut-lines. Each query for a projection point p' of p can be performed by finding the predecessor of the x-coordinate of p' in Q_v (say, an edge of an obstacle o_{pre}) and checking whether p' is inside o_{pre}. If not, then p' becomes a cut-line Steiner point and is associated with the corresponding cut-line. Otherwise, one or two boundary Steiner points are generated for p and are associated with the left and right boundary edges of o_{pre}. Each Steiner point takes $O(1)$ operations on Q_v and O_h, and all boundary Steiner points are generated in a sorted order along their corresponding obstacle boundaries. Thus no sorting is needed.

Therefore, G_1 can be built in $O(n \log n \log \log n)$ time and $O(n \log n)$ space.

3 Finding a Shortest Path in G_1

The generated shortest-path-preserving graph G_1 contains $O(n \log n)$ vertices and edges. If Fredman and Tarjan's shortest path algorithm is applied to G_1, then the time bound will be $O(n \log^2 n)$. To improve this bound, a technique [2, 3, 1] is to reduce the number of vertices of G_1 and increase the number of edges, both by a factor of $\log^{1/2} n$. This is done by partitioning each cut-line, say L, into strips, with each strip containing $O(\log^{1/2} n)$ Steiner points on L. For a particular strip S on L, let $PV_l(S)$ (resp. $PV_r(S)$) be the set of points of U that give rise to the Steiner points in S and that are to the left (resp., right) of L. In order to eliminate the Steiner points in S, except the top and bottom ones in S, it is shown in [2, 3, 1] that it is sufficient to connect each point $p \in PV_l(S)$ to each $q \in PV_r(S)$ with an edge. Thus, in total, $O(\log n)$ edges are generated for S. If we explicitly store those generated edges in each strip, the total space used by G_1 will increase to $O(n \log^{3/2} n)$. To maintain the space bound to $O(n \log n)$, we use the distance table scheme [1] to implicitly represent those generated edges. In table $T(S)$ for S, each point p in $PV_l(S) \cup PV_r(S)$ has an entry which stores the following information: (a) the distance from p to its corresponding steiner point,say p';(b) the distance from p' to the highest steiner point in S, p'_v, which is vertically visible to p'; and (c) the point p'_v. By using this distance table, the weight, or distance, of each generated edge $\overline{pq}$ can be retrieved in constant time.

For the boundary Steiner points, in the full paper, we will show that each obstacle containing boundary Steiner points can be viewed as a "cut-line" and a similar distance table can be maintained for each strip of this obstacle. This main differences are at the structure of this table and the method for computing the weight of the implicitly stored edges.

Theorem 1 *The shortest rectilinear path between two points in a two-layer interconnection model can be computed in $O(n \log^{3/2} n)$ time and $O(n \log n)$ space.*

References

[1] D.Z. Chen, K.S. Klenk and H.-Y.T. Tu, "Shortest path queries among weighted obstacles in the rectilinear plane," *Proc. 11th ACM Symp. Computational Geometry*, 1995, pp. 370–379.

[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, 1989.

[3] D.T. Lee, C.D. Yang and T.H. Chen, "Shortest rectilinear paths among weighted obstacles," *IJCGA*, 1 (2) (1991), pp. 109–124.

[4] D.T. Lee, C.D. Yang and C.K. Wong, "Rectilinear paths among rectilinear obstacles," *Discrete Applied Math.*, 70 (1996), pp. 185–212.

[5] D.T. Lee, C.D. Yang and C.K. Wong, "Finding rectilinear paths among obstacles in a two-layer interconnection model," *IJCGA*, 7 (6) (1997), pp. 581–598.

[6] J.S.B. Mitchell, "Shortest paths among obstacles in the plane," *Proc. 9th ACM Symp. on Comput. Geometry*, 1993, pp. 308–317.

[7] P. van Emde Boas, "Preserving order in a forest in less than logarithmic time and linear space," *Inform. Proc. Letters*, 6 (3) (1977), pp. 80–82.

[8] C.D. Yang, D.T. Lee and C.K. Wong, "Rectilinear path problems among rectilinear obstacles revisited," *SIAM J. Computing*, 24 (3) (1995), pp. 457–472.