Rectilinear Short Path Queries among Rectangular Obstacles

Danny Z. Chen* Kevin S. Klenk†

Abstract

Given a set of n disjoint rectangular obstacles in the plane whose edges are either vertical or horizontal, we consider the problem of processing rectilinear approximate shortest path queries between pairs of arbitrary query points. Our goal is to answer each approximate shortest path query quickly by constructing a data structure that captures path information in the obstacle-scattered plane. We present a data structure for rectilinear approximate shortest path queries that requires $O(n \log^2 n)$ time to construct and $O(n\log n)$ space. This data structure enables us to report the length of an approximate shortest path between two arbitrary query points in $O(\log n)$ time and the actual path in $O(\log n + L)$ time, where L is the number of edges of the output path. If the query points are both obstacle vertices, then the length and an actual path can be reported in $O(1)$ and $O(L)$ time, respectively. The approximation factor for the approximate shortest paths that we compute is 3. The previously best known solution to this problem requires $O(n \log^3 n)$ time and $O(n \log^2 n)$ space to build a data structure, which supports length and actual path queries respectively in $O(\log^2 n)$ and $O(\log^2 n + L)$ time (regardless of the types of query points); the approximation factor for paths between arbitrary query points is 7.

1 Introduction

A *short* path connecting two points p and q in an obstacle-scattered plane is an obstacle-avoiding path whose length is within a small factor c of the length of a *shortest* obstacle-avoiding path connecting p and q in certain metric; such paths are called *c-short* paths. A planar geometric object is *rectilinear* if its boundary edges are either vertical or horizontal. In this paper, we consider the following short path query problem: Given a set of n rectangular obstacles in the plane that are rectilinear and pairwise disjoint, report rectilinear obstacle-avoiding short paths (or their lengths) between pairs of arbitrary query points. Our goal is to answer each short path query quickly by constructing a data structure that captures path information in the obstacle-scattered plane. Clearly, the problem of computing short paths is closely related to the problem of computing shortest paths, which appears in many application areas (such as robotics and VLSI design) and plays vital roles in solving various optimization problems. Shortest obstacle-avoiding path queries often arise in situations where short routes between many pairs of different locations or between two constantly-changing locations are desired. For example, a police car patrolling an area must quickly get to the location where an accident has occurred.

We focus on environments that contain multiple obstacles. Considerable work has been done on computing rectilinear shortest paths among various types of obstacles in the plane (e.g., [1,2,5,7,8,10,11,13–15,17,18,21,22]) and in higher dimensional spaces (e.g., [9,23]). However, very few results are known for answering rectilinear shortest path queries between arbitrary query points. In particular, Atallah and Chen [1] presented a data structure for rectilinear shortest path queries among disjoint rectangular obstacles in the plane; their data structure requires $O(n^2)$ time and space to construct and enables processing a length query in $O(\log n)$ time. ElGindy and Mitra [11] later considered the same problem as [1]; their data structure requires $O(n^{1.5})$ time and space to construct and enables processing a length query in $O(\sqrt{n})$ time. Iwai, Suzuki, and Nishizeki [13] presented a data structure for rectilinear shortest path queries among rectilinear obstacles; the data structure is built in $O(n^2 \log^3 n)$ time and $O(n^2 \log^2 n)$ space, and can be used to answer a length query in $O(\log^2 n)$ time. Very recently, Chen, Klenk, and Tu [5] designed techniques for rectilinear shortest path queries among *weighted* rectilinear polygonal obstacles (resp., arbitrary polygonal obstacles); their data structure requires $O(n^2 \log^2 n)$ time and space to build and enables processing a length query in $O(\log^2 n)$ (resp., $O(\log n)$) time.

*Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, U.S.A. E-mail: chen@cse.nd.edu. Part of this research was done while the author was visiting the Max-Planck-Institut für Informatik in Saarbrücken, Germany.

†Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, U.S.A. E-mail: Kevin.S.Klenk.1@nd.edu.

There is also related work on short path queries in environments with multiple obstacles. Clarkson [6] constructed in $O((n \log n)/\epsilon)$ time a data structure of size $O(n/\epsilon)$, which can be used to process queries of Euclidean $(1+\epsilon)$-short paths in $O(n \log n + n/\epsilon)$ time each, for any given ϵ satisfying $0 < \epsilon \leq 1$. It is possible to extend Clarkson's results such that an $O(n^2 + n/\epsilon)$ space data structure is constructed in $O(n^2 \log n + n^2/\epsilon)$ time; each length query can then be answered in $O((\log n)/\epsilon + 1/\epsilon^2)$ time. Chen [4] recently presented a technique for Euclidean $(6+\epsilon)$-short path queries. Chen's data structure requires $O(n \log n + n/\epsilon)$ space and $O(q^{3/2}/\log^{1/2} q + (n \log n)/\epsilon)$ time to construct, and each length query can be answered in $O((\log n)/\epsilon + 1/\epsilon^2)$ time, where q, $1 \leq q \leq n$, is the minimum number of faces needed to cover all the vertices of some n-vertex planar graph.

Mitra and Bhattacharya [19] first studied rectilinear short path queries among disjoint rectangular obstacles (i.e., the problem of this paper). They built in $O(n \log^3 n)$ time a data structure of $O(n \log^2 n)$ space; using this data structure, the length and an actual short path between any two query points can be reported respectively in $O(\log^2 n)$ and $O(\log^2 n + L)$ time, where L is the number of edges of the output path. The approximation factor for the short paths they computed is 7.[1]

In this paper, we present a data structure for rectilinear short path queries that takes $O(n \log^2 n)$ construction time and $O(n\log n)$ space. This data structure enables us to report the length of a short path between two arbitrary query points in $O(\log n)$ time and an actual path in $O(\log n + L)$ time, where L is the number of edges of the output path. If the query points are both obstacle vertices, then the length and an actual path can be reported in $O(1)$ and $O(L)$ time, respectively. (Note that in [19], the querying time for obstacle vertices is the same as that for arbitrary points.) The approximation factor for the short paths that we compute is 3. Our result makes use of several different data structure techniques and geometric observations.

We will discuss only the algorithms and data structure for reporting the *lengths* of short paths because our solutions can be easily modified for actual short paths in the same complexity bounds.

[1]We learned recently that Mitra and Bhattacharya have improved their approximation factor to 3, within the same time and space complexities as in [19].

2 Preliminaries

The set of n disjoint rectangular obstacles is denoted by R. The vertex set of R is denoted by V_R.

We use p_x and p_y to denote the two coordinates of a point p. A point p is strictly *below* (resp., to the *left* of) a point q iff $p_x = q_x$ and $p_y < q_y$ (resp., $p_y = q_y$ and $p_x < q_x$); we can equivalently say that q is strictly *above* (resp., to the *right* of) p. In the L_1 metric, the *distance* between two points p and q in the plane is $d(p,q) = |p_x - q_x| + |p_y - q_y|$. A line segment with endpoints p and q is denoted by $\overline{pq}$ $(= \overline{qp})$. The *length* of a path C is the sum of the lengths of its constituent segments. $\mathrm{SP}(p,q)$ will denote a *shortest* path between two points p and q. From now on, all paths (shortest or otherwise) are assumed to be obstacle-avoiding.

The *geodesic Voronoi diagram* of points $m_1, m_2, \ldots, m_z$ among the set V_R is a partition of V_R into subsets $H_1, H_2, \ldots, H_z$, such that m_i is associated with H_i, $\bigcup_{i=1}^{z} H_i = V_R$, and for each obstacle vertex $v \in H_i$, v is geodesically nearer to m_i than to any $m_j \neq m_i$.

A path is said to be *monotone* with respect to the x-axis (resp., y-axis) iff its intersection with every vertical (resp., horizontal) line is a contiguous portion of that line. We call a path a *staircase* if it is monotone with respect to both the x- and y-axis. Note that a staircase between two points p and q is a shortest path $\mathrm{SP}(p,q)$ since its length equals $d(p,q)$. A staircase is *unbounded* if it starts and ends with a semi-infinite segment, i.e., a segment that extends to infinity on one side.

The notion of staircase separators was introduced in [1] and is very helpful in the design of divide-and-conquer type algorithms for problems on rectangles. The following lemma from [1] proves the existence and important properties of a staircase separator.

Lemma 1 (Atallah and Chen [1]) *In $O(n \log n)$ time, it is possible to compute an unbounded staircase S, which partitions a set R of n disjoint rectangles into two subsets R_1 and R_2 such that the follow properties hold:*

1. *S does not intersect the interior of any rectangle in R.*
2. *Each of R_1 and R_2 contains no more than $7n/8$ rectangles of R.*
3. *S consists of $O(n)$ line segments.*

The following structures were introduced in [1]. For a point p, the *Northwest* path of p (denoted by the shorthand $\mathrm{NW}(p)$ where 'N' is the mnemonic for

'North' and 'E' for 'East') is the path to infinity obtained by starting at p and going north until reaching an obstacle, at which point we go west along the obstacle's boundary until we clear the obstacle and are able to resume our trip north. One can in this way define an $XY(p)$ path and a $YX(p)$ path for any $X \in \{\text{N}, \text{S}\}$ and $Y \in \{\text{E}, \text{W}\}$ (where 'S' and 'W' are mnemonics for 'South' and 'West' respectively). An $XY(p)$ path starts at p and goes in the X direction whenever it can, and uses a "go in the Y direction" policy for getting around obstacles. A $YX(p)$ is defined similarly.

It has been shown in [1] that the union of the segments of the NW(v) paths, for all $v \in V_R$, forms a *forest* (this we call the *NW-forest*); this forest consists of $O(n)$ vertical and horizontal segments. Furthermore, this forest can be computed in $O(n\log n)$ time [1]. We add a point at infinity as the root for all paths NW(v), $v \in V_R$, thus obtaining from the forest a tree which we call the *NW-tree*. For any point p in the plane, it is easy to show that the unique path NW(p) can be obtained in $O(\log n)$ time from the NW-tree. NW(p) so obtained is represented ***implicitly***; i.e., the path NW(p) consists of a vertical segment, possibly a horizontal segment and a path in the NW-tree. Clearly, NW(p) has $O(n)$ segments. For each $XY \in \{\text{NW}, \text{SE}, \text{SW}, \text{EN}, \text{ES}, \text{WN}, \text{WS}\}$, the XY-forests and XY-trees can be defined similarly.

A point m_v on a staircase separator S is a vertical (resp., horizontal) ***projection point*** of an obstacle vertex v if and only if $\overline{m_v v}$ is a vertical (resp., horizontal) line segment that does not intersect the interior of any obstacle. Let M be the set of vertical and horizontal projection points of V_R on S. The following lemma, proved in [1] and also in [19], shows how a staircase separator controls certain shortest paths between opposite sides of the separator.

Lemma 2 *Let S be a staircase separator that divides the set R of rectangular obstacles into two subsets R_1 and R_2. Then for any obstacle vertices $h_i \in V_{R_1}$ and $h_j \in V_{R_2}$, there exists a shortest path* SP(h_i, h_j) *which contains a horizontal or vertical projection point of M.*

Consider two arbitrary obstacle vertices h_i and h_j of R which lie on opposite sides of the staircase separator S. Among all the points of M, let $m_k, m_l \in M$ be the geodesically nearest points of h_i and h_j respectively. Note that SP(m_k, m_l) can be taken along the staircase S. The following lemma, due to Mitra and Bhattacharya [19], is a key to computing short paths.

Lemma 3 (Mitra and Bhattacharya [19]) *The length of the path $Q =$ SP$(h_i, m_k) \cup$ SP$(m_k, m_l) \cup$ SP(m_l, h_j) between h_i and h_j is at most three times the length of a shortest path* SP(h_i, h_j).

3 Construction of the Data Structure

Our data structure for rectilinear short path queries consists of several key components: (1) two planar point-location structures that allow us to perform ray-shooting operations in the vertical and horizontal directions; (2) a structure that ***implicitly*** maintains the length of a short path between every pair of obstacle vertices in V_R; (3) the XY-trees as described in Section 2 and additional structures for certain operations on these trees. The two planar point-location structures are taken from the data structure of [1] and can be constructed in $O(n\log n)$ time and $O(n)$ space. Hence our discussion focuses on the second and third components of the data structure.

Mitra and Bhattacharya [19] presented a data structure for implicitly maintaining rectilinear short paths between obstacle vertices. They used the following algorithm to build their data structure.

1. Compute the staircase separator S that partitions R into two subsets R_1 and R_2 (Lemma 1).
2. Compute the set M of $O(n)$ vertical and horizontal projection points of V_R on S.
3. Construct the geodesic Voronoi diagram of M among V_R.
4. Perform this procedure recursively on R_1 and R_2 until each subset contains one obstacle.

Mitra and Bhattacharya implemented the above algorithm in $O(n \log^3 n)$ time and $O(n \log^2 n)$ space.

To construct our structure for implicitly maintaining lengths of short paths between obstacle vertices, we modify the above algorithm of Mitra and Bhattacharya [19]. Note that both Steps 1 and 2 can be performed in $O(n\log n)$ time and $O(n)$ space [1]. Step 3 can be done also in $O(n\log n)$ time and $O(n)$ space by simply using Mitchell's algorithm [17,18] for computing the geodesic Voronoi diagram of a set of sites in the L_1 plane with rectilinear obstacles.

Let T be the recursion tree for the modified algorithm; that is, the root of T corresponds to the "top-level" recursive call (the one associated with R), the children of the root correspond to the recursive calls for R_1 and R_2, and so on. It is easy to further modify the algorithm so that the information (separators, geodesic Voronoi diagrams, etc) produced by each recursive call remains stored in T even after that call returns. We assume that this modification has already been done, so that each node v of T stores the vertices of the obstacle set $R_v \subseteq R$ associated with v, the staircase separator S_v partitioning R_v, and the geodesic Voronoi diagram of M_v (on S_v) among V_{R_v}.

Lemma 4 *The recursion tree T can be constructed in $O(n \log^2 n)$ time and $O(n \log n)$ space.*

Proof: It is easy to see that the recurrence relation $T(n)$ for the time complexity of the algorithm is $T(n) \leq T(an) + T((1-a)n) + bn\log n$, where $1/8 \leq a \leq 7/8$ and b is some positive constant; thus $T(n) = O(n\log^2 n)$. The space taken by the tree T and all the information stored with its nodes obeys the recurrence relation $S(n) \leq S(an) + S((1-a)n) + cn$ for $1/8 \leq a \leq 7/8$ and some positive constant c; thus $S(n) = O(n \log n)$. □

Processing a length query between two obstacle vertices requires $O(\log^2 n)$ time in [19]. We take a different approach for length queries between obstacle vertices and handle each such query in $O(1)$ time. To achieve this goal, we need to preprocess the recursion tree T so that we can quickly answer the following kind of queries: "Given any two nodes $a, b \in T$, return the lowest common ancestor of a and b in T." It is well-known that the lowest common ancestor queries in a tree can be answered by preprocessing the tree in $O(n)$ time and space, so that each query takes $O(1)$ time [12, 20]. This completes our discussion of the second component of our short path data structure.

For the third component of our short path data structure, we build all the XY-trees (Section 2) as described in [1]. However, the data structure in [1] for the XY-forests stores the XY-path from each obstacle vertex in V_R to the point at infinity *explicitly*; this storage required $O(n^2)$ space (but this did not affect the overall efficiency of the algorithm [1]). Atallah and Chen [1] used this explicit representation of XY-forests so that they could perform binary searches on the XY-path of each vertex in V_R. We would still like to be able to perform binary searches on the XY-paths of obstacle vertices; however, we are not willing to pay the space penalty as in [1]. Towards this goal, we preprocess each XY-tree Z so that the following type of queries can be quickly answered: "Given a vertex v in Z and a positive integer i, find the i-th vertex on the path in Z from v to the root of Z." Such queries are called *level-ancestor queries* by Berkman and Vishkin [3], who gave a linear time algorithm for preprocessing rooted trees so that the level-ancestor queries can be answered in $O(1)$ time each. This preprocessing allows us to perform the same binary searches on XY-paths that are implicitly represented in XY-trees without the space penalty caused by using a separate representation for each XY-path. The details of our querying procedures will be given in the next section.

The result of this section is summarized in the following theorem.

Theorem 1 *The data structure for rectilinear approximate shortest path queries among n disjoint rectangular obstacles can be constructed in $O(n\log^2 n)$ time and $O(n \log n)$ space, with an approximation factor of* 3.

4 Short Path Queries

Our idea for computing a short path between two arbitrary query points is to reduce such a query to queries between $O(1)$ vertices in V_R. In order to do this, we will first present a method for performing length queries between vertices in V_R (henceforth called "V_R-Queries"), which can be performed in $O(1)$ time each (an improvement of a $\log^2 n$ factor over the previously best known result [19]). We will next show how an arbitrary query can be reduced to $O(1)$ V_R-Queries in $O(\log n)$ time.

4.1 V_R-Queries

The method for answering a V_R-Query between two vertices h_i and h_j in V_R is as follows:

1. Determine the node v in the recursion tree T such that the staircase separator S_v associated with v splits for the first time h_i and h_j into opposite sides of the separator (i.e., for every proper ancestor u of v in T, h_i and h_j belong to the same side of the separator S_u).

2. Use the geodesic Voronoi diagram stored in the node v of T to report the length of a short path between h_i and h_j based on Lemma 3.

Note that the above procedure, V_R-QUERY, determines S_v (Step 1) in a completely different manner from [19]. In [19], S_v was identified by performing $O(\log n)$ binary searches along a path in their recursion tree, one at each level of the tree. We simply use the observation that S_v is stored at the lowest common ancestor of the leaves that store h_i and h_j in T. Therefore, after preprocessing the recursion tree T in $O(n)$ time and space, we can find the correct S_v in $O(1)$ time by using a lowest common ancestor query [12,20]. From Lemma 3, we know the approximation factor of the short path that we compute is 3. Thus we have the following lemma.

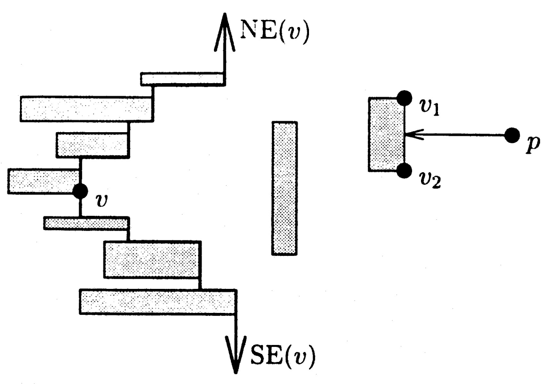

Figure 1: A leftward ray-shooting from an arbitrary point p.

Lemma 5 *The procedure V_R-*Query *computes the length of an approximate shortest path between any two vertices h_i and h_j of V_R in $O(1)$ time, with an approximation factor of* 3.

4.2 Queries between Two Arbitrary Points

In this subsection, we relax the restriction of both the query points being vertices in V_R. Given the procedure for V_R-Queries, we are able to handle a length query between two *arbitrary* points in the plane in $O(\log n)$ time. Our solution for length queries between arbitrary points is based on several geometric observations. We begin with the case of queries with only one arbitrary query point, the other query point being in V_R, and then we later extend it to the case of two arbitrary query points. The approximation factor that we obtain is 3.

Suppose we want to compute the length of a short path between an arbitrary point p and an obstacle vertex v. WLOG, assume p is in the first quarter of v (the other cases can be handled similarly). Then the procedure, called One-Arbitrary-Query, for such a query is as follows:

1. Let L be the horizontal line passing p. Compute the intersection of L and NE(v). This can be done by performing a binary search for p_y on the path NE(v).

2. If p is to the right (resp., left) of $L \cap \text{NE}(v)$, then based on the observations of [10], a *shortest* p-to-v path is monotone with respect to the x-axis (resp., y-axis). WLOG, assume that p is to the right of $L \cap \text{NE}(v)$ (see Figure 1).

3. Perform a ray-shooting operation: The ray starts at p and goes horizontally to the left of p. If the ray intersects NE(v) before it hits an obstacle, then a *shortest* p-to-v path is found (first from p to $L \cap \text{NE}(v)$ and then along NE(v) to v); otherwise, the ray hits an obstacle edge without crossing NE(v). Let v_1 and v_2 be the vertices of that obstacle edge (Figure 1).

4. Based on the observations of [1, 10], a *shortest* p-to-v path must go through one of v_1 and v_2. Hence the length of a *short* p-to-v path can be taken as the minimum of the length of a short v_1-to-v path plus $d(p, v_1)$ and the length of a short v_2-to-v path plus $d(p, v_2)$. The queries on short v_1-to-v and v_2-to-v paths are clearly V_R-Queries.

We first show the claim that the length of the short p-to-v path so computed is within a factor 3 of the length of a *shortest* p-to-v path. WLOG, assume a shortest p-to-v path goes through v_1. Note that the length of the short p-to-v path that we compute cannot be longer than the length of a short v_1-to-v path plus $d(p, v_1)$, and that there is no approximation in the distance $d(p, v_1)$. But the length of the short v_1-to-v path that we compute is within a factor 3 of the length of a shortest v_1-to-v path (by Lemma 5). Hence the claim follows.

We now show that the procedure One-Arbitrary-Query can be performed in $O(\log n)$ time. Given $L \cap \text{NE}(v)$, Step 2 can be easily done in $O(1)$ time. Step 4 also takes $O(1)$ time based on Lemma 5. The ray-shooting in Step 3 requires $O(\log n)$ time by using the horizontal planar point-location structure (i.e., the first component of our short path data structure). Hence the key is Step 1 that finds $L \cap \text{NE}(v)$. To obtain $L \cap \text{NE}(v)$ in $O(\log n)$ time, we perform a binary search on the path NE(v). It is easily seen that the level-ancestor queries on the NE-tree enable us to perform an $O(\log n)$ time binary search on NE(v). This is because by using a level-ancestor query, we can access in $O(1)$ time any ancestor of v in the NE-tree, without having the path NE(v) stored explicitly in a separate array.

A length query between two arbitrary points p and q is reduced in $O(\log n)$ time to two length queries, each of which is between one arbitrary point and one obstacle vertex, as follows. WLOG, assume p is in the first quarter of q. We first compute the path NE(q). This can be obtained in $O(\log n)$ time by performing an upward ray-shooting from q and then taking (implicitly) the NE(u) path from the NE-tree, where u is the right vertex of the obstacle edge whose interior is hit by the upward ray from q. Next, by using a procedure similar to the procedure One-Arbitrary-Query, we reduce the computation of a short p-to-q path to computing two short q-to-v_1 and q-to-v_2 paths, each of which can then be handled by the procedure ONE-ARBITRARY-QUERY, with both v_1 and v_2 being obstacle vertices. The time analysis and correctness proof of this query procedure are similar to those for the procedure ONE-ARBITRARY-QUERY.

Our result of this subsection is summarized in the following theorem.

Theorem 2 *Each query for the length of an approximate shortest path between two arbitrary points can be processed in $O(\log n)$ time, with an approximation factor of* 3.

References

[1] M. J. Atallah and D. Z. Chen. "Parallel rectilinear shortest paths with rectangular obstacles," *Computational Geometry: Theory and Applications*, 1 (1991), pp. 79–113.

[2] M. J. Atallah and D. Z. Chen. "On parallel rectilinear obstacle-avoiding paths," *Computational Geometry: Theory and Applications*, 3 (1993), pp. 307–313.

[3] O. Berkman and U. Vishkin. "Finding level-ancestors in trees," Tech. Rept. UMIACS-TR-91-9, University of Maryland, 1991.

[4] D. Z. Chen. "On the all-pairs Euclidean short path problem," *Proc. of the Sixth Annual ACM-SIAM Symp. on Discrete Algorithms*, San Francisco, January 1995, pp. 292–301.

[5] D. Z. Chen, K. S. Klenk, and H.-Y. T. Tu. "Shortest path queries among weighted obstacles in the rectilinear plane," to appear in the *Eleventh Annual ACM Symp. on Computational Geometry*, Vancouver, Canada, June 1995.

[6] K. L. Clarkson. "Approximation algorithms for shortest path motion planning," *Proc. 19th Annual ACM Symp. Theory of Computing*, 1987, pp. 56–65.

[7] K. L. Clarkson, S. Kapoor, and P. M. Vaidya. "Rectilinear shortest paths through polygonal obstacles in $O(n(\log n)^2)$ time," *Proc. 3rd Symp. on Computational Geometry*, 1987, pp. 251–257.

[8] K. L. Clarkson, S. Kapoor, and P. M. Vaidya. "Rectilinear shortest paths through polygonal obstacles in $O(n \log^{3/2} n)$ time," submitted for publication.

[9] M. de Berg, M. van Kreveld, and B. J. Nilsson. "Shortest path queries in rectangular worlds of higher dimension," *Proc. 7th Annual ACM Symp. on Computational Geometry*, 1991, pp. 51–59.

[10] P. J. de Rezende, D. T. Lee, and Y. F. Wu. "Rectilinear shortest paths in the presence of rectangles barriers," *Discrete Comput. Geom.*, 4 (1989), pp. 41–53.

[11] H. ElGindy and P. Mitra. "Orthogonal shortest route queries among axes parallel rectangular obstacles," *Int. J. of Comput. Geom. and Appl.*, 4 (1) (1994), pp. 3–24.

[12] D. Harel and R. E. Tarjan. "Fast Algorithms for finding nearest common ancestors," *SIAM J. Comput.*, 13 (1984), pp. 338–355.

[13] M. Iwai, H. Suzuki, and T. Nishizeki. "Shortest path algorithm in the plane with rectilinear polygonal obstacles" (in Japanese). *Proc. of SIGAL Workshop*, Ryukoku University, Japan, July 1994.

[14] R. C. Larson and V. O. Li. "Finding minimum rectilinear distance paths in the presence of barriers." *Networks*, 11 (1981), pp. 285–304.

[15] D. T. Lee, T. H. Chen, and C. D. Yang. "Shortest rectilinear paths among weighted obstacles." *International Journal of Computational Geometry and Applications*, 1 (2) (1991), pp. 109–124.

[16] K. Mehlhorn. "A faster approximation algorithm for the Steiner problem in graphs," *Inform. Process. Lett.*, 27 (1988), pp. 125–128.

[17] J. S. B. Mitchell. "L_1 shortest paths among polygonal obstacle in the plane," *Algorithmica*, 8 (1992), pp. 55–88.

[18] J. S. B. Mitchell. "An optimal algorithm for shortest rectilinear path among obstacles." *First Canadian Conference on Computational Geometry*, 1989.

[19] P. Mitra and B. Bhattacharya. "Efficient approximation shortest-path queries among isothetic rectangular obstacles," *Proc. 3rd Workshop on Algorithms and Data Structures* (WADS'93), 1993, pp. 518–529.

[20] B. Schieber and U. Vishkin. "On finding lowest common ancestors: Simplification and parallelization," *SIAM J. Comput.*, 17 (1988), pp. 1253–1262.

[21] 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. on Computers* C-36 (1987), 321–331.

[22] C. D. Yang, T. H. Chen, and D. T. Lee. "Shortest rectilinear paths among weighted rectangles." *Journal of Information Processing*, 13 (4) (1990), pp. 456–462.

[23] C.-K. Yap and J. Choi. "Rectilinear geodesics in 3-space," to appear in the *Eleventh Annual ACM Symp. on Computational Geometry*, Vancouver, Canada, June 1995.