Geometric Path Problems with Violations[*]

Anil Maheshwari[1], Subhas C. Nandy[2], Drimit Pattanayak[3], Sasanka Roy[2] and Michiel Smid[1]

[1]School of Computer Science, Carleton University, Ottawa, Canada, {anil,michiel}@scs.carleton.ca.
[2]Indian Statistical Institute, Kolkata, India, {nandysc,sasanka}@isical.ac.in
[3]Chennai Mathematical Institute, Chennai, India, drimit@cmi.ac.in

Abstract

In this paper, we study variants of the classical geometric shortest path problem inside a simple polygon, where we allow a part of the path to go outside the polygon. Let P be a simple polygon consisting of n vertices and let s, t be a pair of points in P. Let $\text{int}(P)$ represent the interior of P and let $\overline{P}$ represent the exterior of P, i.e. $\text{int}(P) = P \setminus \partial(P)$ and $\overline{P} = \mathbb{R}^2 \setminus \text{int}(P)$. For an integer $k \geq 0$, we define a *k-violation path* from s to t to be a path connecting s and t such that its intersection with $\overline{P}$ consists of at most k segments. There is no restriction in terms of the number of segments of the path within P. The objective is to compute a path of minimum Euclidean length among all possible $(\leq k)$-violation paths from s to t. In this paper, we study this problem for $k = 1$ and propose an algorithm that computes the shortest one-violation path in $O(n^3)$ time. We show that for rectilinear polygons, the minimum length *rectilinear one-violation path* can be computed in $O(n \log n)$ time.

We extend the concept of *one-violation path* to a *one-stretch violation path*. In this case, the path between s and t is composed of (a) a path in P from s to a vertex u of P, (b) a path in $\overline{P}$ between u and a vertex v of P, and (c) a path in P between v and t. We show that a minimum length one-stretch violation path can be computed in $O(n \log n \log \log n)$ time.

Next, we introduce *one- and two-violation monotone rectilinear path problems* among a set of n disjoint axis-parallel rectangular objects. Let s, t be two points in $\mathbb{R}^2$ that are not in the interior of any of the objects. In the case of one-violation monotone path problem, the desired rectilinear path from s to t consists of horizontal edges that are directed towards the right and vertical edges that are directed upwards, except for at most one edge. Similarly, in the case of a two-violation monotone path problem, all horizontal edges are directed towards the right except at most one and all vertical edges are directed upwards except at most one. Our algorithms for both of these problems runs in $O(n \log n)$ time.

Keywords: shortest path, violations, simple polygons, rectilinear polygons, graph, geometry

[*]Research supported by NSERC and the Canadian Commonwealth Scholarship.

1 Introduction

Let P be a simple polygon consisting of n vertices and let s, t be a pair of points in P. Let $\text{int}(P)$ represent the interior of P and let $\overline{P}$ represent the exterior of P, i.e. $\text{int}(P) = P \setminus \partial(P)$ and $\overline{P} = \mathbb{R}^2 \setminus \text{int}(P)$.

In the traditional shortest path problem inside a simple polygon, the input consists of P and a pair of points $s, t \in P$; the objective is to connect s and t by a path in P of minimum length. Here a path is a sequence of line segments, called the *edges* of the path; the path changes the direction (or *turns*) only at the vertices of P. The *length* of a path is the sum of lengths of all the edges on that path. The shortest path problem inside a simple polygon has a long and rich history. Interested readers may see [8, 17, 19] for an exhaustive survey of known results. It is well-known that the shortest path map from a point s to all the vertices in P can be computed in $O(n)$ time [11].

In this paper, we study variants of the geometric shortest path problem where we allow a part of the path to go outside P. For an integer $k \geq 0$, we define a *k-violation path* between s and t to be a path connecting s and t such that its intersection with $\overline{P}$ consists of at most k segments. There is no restriction in terms of the number of segments of the path within P. The objective is to compute the path of minimum Euclidean length among all possible ($\leq k$)-violation paths from s to t. We study the following variants:

One-Violation Path Problem: Given a simple polygon P and a pair of points $s, t \in P$, compute a shortest one-violation path between s and t. Note that the intersection of a one-violation path with $\overline{P}$ is at most one segment (see Figure 1 for an illustration).

One-Stretch Violation Path Problem: Given a simple polygon P and a pair of points $s, t \in P$, a *one-stretch path* between s and t is composed of (a) a path in P from s to a vertex u of P, (b) a path in $\overline{P}$ between u and a vertex v of P, and (c) a path in P between v and t. The one-stretch violation path problem is to compute a shortest path among all one-stretch paths between s and t.

Monotone Rectilinear Path Problems with Violations: Given a set $\mathcal{R}$ of disjoint axis-parallel rectangular obstacles in $\mathbb{R}^2$, and a pair of points $s, t \in \mathbb{R}^2 \setminus \mathcal{R}$, a monotone rectilinear path from s to t consists of horizontal edges that are directed towards the right and vertical edges that are directed upwards. A monotone rectilinear path from s to t may not always exist. We show that if we allow at most one-violation with respect to the direction of a horizontal or a vertical edge, then a path from s to t always exists. The objective in these problems is to compute a minimum length monotone rectilinear path between s and t with at most one or two violation edges.

In computational geometry, it is customary to study classical optimization problems that violate a given set of constraints in some restricted way, see e.g. [4, 18, 21]. Also, there are several studies in finding shortcuts for a given geometric network (paths, trees, cyclic, and plane networks) so as to reduce its diameter or the spanning ratio, see e.g. [3, 6, 9]. An alternative formulation of the one-violation path problem is to find a shortcut for a path between s and t in a simple polygon P, subject to the condition that the intersection of the shortcut with $\overline{P}$ is at most one segment. These problems and their variants motivated us to seek algorithms for the traditional geometric shortest path problems with violations.

1.1 New Results

In this paper, we propose algorithms for the following variants of the one-violation path problem:

- An algorithm for computing a shortest *one-violation path* between a pair of points inside a simple polygon with n vertices; it runs in $O(n^3)$ time.
- An algorithm for computing a shortest *one-violation rectilinear path* between a pair of points inside a rectilinear polygon; it runs in $O(n \log n)$ time.
- Extending one-violation path problem to *one-stretch violation path* among a pair of points inside a simple polygon. The time complexity of our proposed algorithm for this problem is $O(n \log n \log \log n)$.
- For a given set $\mathcal{R}$ of disjoint axis-parallel rectangular obstacles and an arbitrary pair of points s and t in $\mathbb{R}^2 \setminus R$, we show that a monotone rectilinear path from s to t with at most one (resp. two, where one is in horizontal direction and the other one is in vertical direction) violation(s) always exists, and a path of minimum length among such paths can be computed in $O(n \log n)$ time.

1.2 Organization

The paper is organized as follows. In Section 2, we introduce the shortest path problem with violations in a graph with two types of edges, namely *good* and *bad*. We show that Dijkstra's shortest path algorithm can be modified to work for this problem. In Section 3, the one-violation shortest path problem in a simple polygon is studied. In Section 4, we show that for rectilinear polygons, the time complexity of the one-violation shortest path problem can be improved. In Section 5, we extend the idea of one-violation shortest path to one-stretch violation shortest path for simple polygons, and provide an algorithm with improved time complexity. In Section 6, we study the one/two violation rectilinear monotone shortest path problem among a set of disjoint rectangular obstacles. Finally, concluding remarks appear in Section 7.

2 Shortest path with violations in a graph

We first introduce the k-violation shortest path problem in a simple graph $G = (V, E = E_1 \cup E_2)$, where we have two disjoint sets of edges E_1 and E_2, named *good* and *bad* respectively. Each edge in $(u, v) \in E_1 \cup E_2$ is assigned with a non-negative cost $c(u, v)$. We can compute a path from s to t by choosing as many good edges as required but are allowed to use at most k *bad* edges to reduce the cost of the path. The objective is to have a path of minimum cost from s to t. We show that Dijkstra's algorithm can be tailored to work for this problem using Fibonacci heaps[1] [7] as the additional data structure. Here

[1]Each element of the heap is a tuple (a, b, x), where a is the key and b is the priority field. Needless to say, each key $a \in A = \{a_1, a_2, \ldots, a_N\}$ has at most one representation in the heap. Each element in the heap may contain some other information x, which can be null also.

Algorithm 1: Dijkstra-k-Violation-Algorithm(G, s, t)

```
Input: The weighted digraph $G = (V, E)$. An integer $k \geq 0$.
Each edge $(u, v)$ has a weight $c(u, v)$ and a flag $f(u, v)$ indicating "good" or "bad".
Output: The $k$-violation shortest path from $s$ to $t$
1  insert$((s, 0, 0), H)$                         // Initialize the element of $H$
2  forall $v \in V \setminus \{s\}$ do
3      forall $\mu = 0, 1, 2, \ldots, k$ do
4          insert$((v, \mu, \infty), H)$
5  repeat
6      $(u, \mu, \chi)$ = delete-min$(H)$ // Delete the root element from $H$
7      if $u = t$ then
8          Report $\chi$ and Exit;
9      else
10         forall $v \in Adj(u)$                  //$Adj(u)$: adjacency list of node $u$
11          do
12             (* Process the edge $(u, v)$ *)
13             if $f(u, v)$ = "good" then
14                 Create a tuple $(v, \mu', \chi') = (v, \mu, \chi + c(u, v))$
15             if $f(u, v)$ = "bad" then
16                 Create a tuple $(v, \mu', \chi') = (v, \mu + 1, \chi + c(u, v))$
17             if $\mu' \leq k$ then
18                 decrease-key$((v, \mu', \chi'), H)$
19 until $H$ is empty;
```

insert[2], decrease-key[3], find-min[4] operations can be performed in $O(1)$ time, and delete-min[5] operation needs logarithmic time on the size of the heap.

During the execution of the algorithm, we use a priority queue H maintained as a Fibonacci heap. Each node of H is a triple (v, μ, χ) that corresponds to a path from s to the node v with violation count μ; χ denotes the length of this path. Here (v, μ) plays the role of key and χ plays the role of priority in the Fibonacci heap H. In addition, we maintain k pointers C_v (in an array) with each node $v \in V$. The μ-th entry of C_v stores the position of key value (v, μ) in H. The entries of $\{C_v, v \in V\}$ are updated during the insert, decrease-key and delete-min operations in H. Initially, the heap H contains $n(k+1)$ elements for $n(k+1)$ possible key values $\{(v, \mu) | v \in V, \mu = 0, 1, 2, \ldots, k\}$. The χ value of each element is set to ∞. The locations $C_v, v \in V$ are set accordingly.

In each iteration of the modified Dijkstra's algorithm, we choose an element of H having minimum cost χ (say) by invoking delete-min(H). Let it correspond to the node $u \in V$. We relax node u, or in other words, process the edges (u, v) for all vertices in V adjacent to u. For each edge (u, v), it generates a tuple (v, μ', χ'), where $\chi' = \chi + c(u, v)$ and

[2]insert(a, b, x) inserts a tuple in the heap H.

[3]decrease-key(a, b') updates the b-value corresponding to the key a in the heap H if the existing b-value of the node having key a is greater than b'; if the update is done, then it adjusts the heap H.

[4]find-min(H) returns the entry with minimum priority (b) in the heap H.

[5]delete-min(H) deletes the element with minimum priority from H, and then adjusts the heap H.

$\mu' = \mu$ or $\mu + 1$ depending on whether the edge (u, v) is “good“ or “bad”. If $\mu' \leq k$ and χ' is less than the χ value attached to the (v, μ')-th node of H[6], we invoke decrease-key((v, μ', χ'). The algorithm terminates when t is reached for the first time, or in other words, when a tuple (v, μ, χ) is taken from the heap with $v = t$. The pseudo code of the proposed method is given in Algorithm 1.

Theorem 1. *The Dijkstra-k-Violation-Algorithm correctly computes a path of minimum cost consisting of at most k violation edges in $O(mk + nk \log n)$ time using $O(m + nk)$ space, where $n = |V|$ and $m = |E_1| + |E_2|$.*

Proof. The correctness of the algorithm follows from that of the Dijkstra's shortest path algorithm, and the fact that we are using the tuple (v, μ) as the key in the priority queue H.

The first part of the time complexity follows from the fact that (i) each edge may need to be processed at most k times, and we are using Fibonacci heap for implementing the priority queue. The second part of the time complexity follows from the fact that a node of the graph may be considered for relaxing at most k times with k different violation counts. The delete-min operation needs to be invoked $O(nk)$ times, and each delete-min operation needs $O(\log(nk))$ time, where nk is the size of H.

Apart from storing the graph, which needs $O(\max(m, n))$ space, the space needed for storing C_v for all $v \in V$ is $O(nk)$. Also the priority queue H needs $O(nk)$ space. Thus, the space complexity follows. □

3 One-violation path problem in a simple polygon

We are given a simple polygon P consisting of n vertices, and a pair of points $s, t \in P$. The objective is to compute a minimum length path from s to t that consists of at most one edge of the path or its portion that passes through the outside of the polygon P. From now onwards, we use $\overline{P} = \mathbb{R} \setminus \text{int}(P)$ to denote the region outside the polygon P, and the edge of a path from s to t that passes through $\overline{P}$ is referred to as the *violation edge*. Note that, unlike the shortest path inside a simple polygon, here a bend in a path may not always be at a vertex of the polygon (see Figure 1). It is possible that the *violation edge* of the shortest path from s to t meets at a point in the interior of an edge of the polygon, from where the path enters in (leaves from) the polygon.

Throughout the paper, “$x \rightsquigarrow y$” is used to indicate a polyline path from x to y, and “$x \longrightarrow y$” to indicate an edge of the path. We use $\Pi_{in}(x, y)$ to denote the shortest path between a pair of points $x, y \in P$ inside the polygon.

Observation 1. *Unless the line segment st lies completely inside P, there exists a one-violation path from s to t that is shorter than $\Pi_{in}(s, t)$.*

Proof. Since the line segment st does not lie completely inside P, there exists at least one turn at a vertex, say v, of $\Pi_{in}(s, t)$. We choose two points α and β at a very small distance

[6]The μ'-th entry of C_v indicates this position.

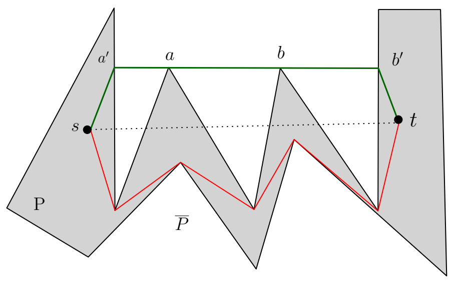


Figure 1: Illustration of $\Pi_{in}(s,t)$ (red path) and the shortest one-violation path between s and t (green path). The shortest one-violation path uses segment ab and bends at a' and b'. Points a' and b' are in the interior of edges. The violation edge $a'b'$ passes through the vertices a and b.

ϵ from v on the two segments of $\Pi_{in}(s,t)$ adjacent to v such that the segment $\alpha\beta \cap \overline{P}$ forms one connected component (i.e., $\alpha\beta$ is a violation edge). By triangle inequality, the path obtained by replacing the path segment $\alpha \to v \to \beta$ of $\Pi_{in}(s,t)$ by $\alpha \to \beta$ is shorter, and gives a feasible one-violation path from s to t. □

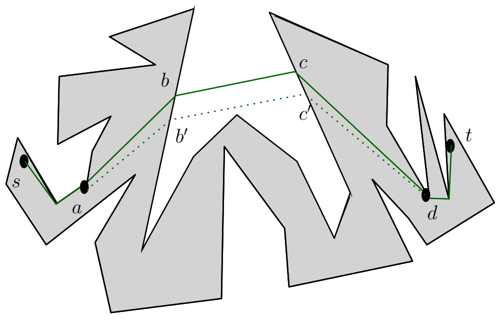
Lemma 1. *Let $\Pi = s \rightsquigarrow a \to b \to c \to d \rightsquigarrow t$ be any optimum one-violation path from s to t consisting of at least three segments, with bc as the violation edge and a and d are vertices of P and b and c are points on the boundary of P. Then, the violation edge bc either (i) passes through a vertex of P or (ii) points $\{a,b,c\}$ or $\{b,c,d\}$ are collinear.*

Proof. We prove this by contradiction. Let $\Pi = s \rightsquigarrow a \to b \to c \to d \rightsquigarrow t$ be an optimum one-violation path where the violation edge bc does not pass through any vertex of P and neither $\{a,b,c\}$ nor $\{b,c,d\}$ are collinear. Depending on the slope of the edges ab and cd of Π, we need to consider two cases as shown in Figures 2(a) and 2(b).

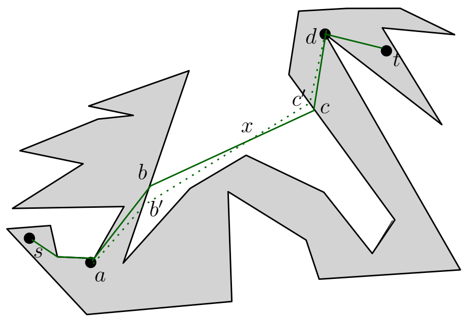
Consider first the case that the edges ab and cd of Π have slopes of different sign (i.e., ab, bc, and cd form a convex path). In this case, the violation edge bc can be moved to a new position $b'c'$, parallel to bc (see Figure 2(a)), such that $b'c'$ is a violation edge and the new path $\Pi' = s \rightsquigarrow a \to b' \to c' \to d \rightsquigarrow t$ is shorter than Π. This contradicts the minimality of Π. Note that we can continue this movement until bc either (i) touches a vertex of P, or (ii) b, c, and a and/or d becomes collinear.

Now consider that the edges ab and cd of Π have slopes of the same sign. Choose a point x on bc. Rotate bc around x so that the new segment $b'c'$ is a violation edge and the path $\Pi' = s \rightsquigarrow a \to b' \to c' \to d \rightsquigarrow t$ is a valid one violation path (see Figure 2b). We claim that we can always rotate bc such that the length of Π' is less than the length of Π. This follows from the triangle inequality as the path segment $a \to b' \to x \to c' \to d$ is shorter than $a \to b \to c \to d$. Hence this contradicts the minimality of Π. Note that we can rotate bc around x until the line segment $b'c'$ either (i) touches a vertex of P or (ii) a, b', c', and d becomes collinear. □

We will use the shortest path maps of s and t in P, namely SPM_s and SPM_t, respectively, to compute an optimum one-violation path. Shortest path maps are typically used for



(a)



(b)

Figure 2: Illustration of the proof of Lemma 1

computing the shortest path of all points in P from s that stays within P. These maps split the boundary of the polygon into open intervals, called *component-pieces*. None of the component-pieces contain any vertex of P.

Definition 1. *Each component-piece is associated with a vertex of the polygon, called the parent vertex, such that for every point in that component piece, the shortest path from s is reached from its parent vertex.*

SPM_s can be visualized as a tree. Its root is s, and component-pieces are leaves. The vertices of P may appear as both leaf or non-leaf nodes in the tree. Each node p of SPM_s, that is a vertex of P, stores the length of the shortest path from s to p inside the polygon. A similar data structure is also prepared for SPM_t.

Lemma 2. *[11, 13] After a linear time preprocessing, the length of the shortest paths $\Pi_{in}(s, p)$ and $\Pi_{in}(t, p)$ for any point p on a component-piece can be computed in $O(1)$ time.*

We consider the convex hull $CH(P)$ of P. Each edge of $CH(P)$, that is not an edge of P, introduces a pocket with respect to P as defined below.

Definition 2. *A* pocket *of a simple polygon P is defined by an edge e of $CH(P)$ that is not an edge of P. It is a polygonal region outside P but inside $CH(P)$. It is bounded by e and a sequence of consecutive edges of P where the first and the last edge of the sequence are incident on the two end points of e. The two end points of e will be referred to as the* frontiers *of the pocket.*

Let us name these pockets as $P_1, P_2, \ldots, P_k$. Each pocket along with the corresponding edge of $CH(P)$ defines a simple polygon outside P. Let $\Pi = s \rightsquigarrow a \to b \to c \to d \rightsquigarrow t$ be a one-violation path consisting of at least three segments as stated in Lemma 1. In this notation, a and d are vertices of P, bc is the violation edge, where b and c are points on the boundary of P. Let b belongs to the component piece I of SPM_s and let c belongs to the component piece J of SPM_t. Note that the parent of I in SPM_s is a and the parent of J in SPM_t is d. Observe that Π is composed of up to five pieces, namely (i) $\Pi_{in}(s, a)$, (ii) $\Pi_{in}(t, d)$, (iii) the segment bc lying completely within one of the pockets P_i, where b

and c are on the boundary of P, (iv) a segment ab in P, and (v) a segment cd in P. It is possible that the segments ab and bc may be collinear and/or the segments bc and cd may be collinear.

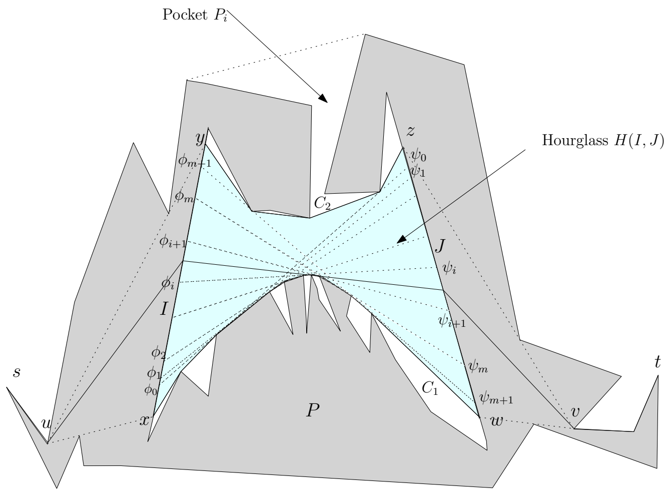


Figure 3: Processing of a pair of component-pieces (I, J)

Let $I = [xy]$ and $J = [wz]$ be two component pieces of a pocket P_i, with their parent vertices u and v with respect to SPM_s and SPM_t, respectively. We want to compute an optimum one-violation path between s and t, where the violation edge bc is restricted to $b \in I$ and $c \in J$. Assume that in the traversal of the boundary of P_i in counterclockwise order starting at y, the points x, w, and z appear in this order (see Figure 3). Consider the shortest paths between x and w and between y and z restricted to lie within P_i. These two shortest paths together with the component pieces I and J define an hourglass $H(I, J)$ in P_i [10, 11]. An hourglass is said to be *open*, if the corresponding shortest paths do not share any vertex. Otherwise it is *closed*. We make the following observation based on the well established connection between shortest paths and hourglasses (see [10, 11]).

Observation 2. *For two component pieces I and J of a pocket P_i, there exists a segment joining I and J that lies completely within P_i if and only if $H(I, J)$ is open.*

For a pair of component pieces I and J in a pocket P_i, we first check whether $H(I, J)$ is open by computing the corresponding shortest paths in P_i. If $H(I, J)$ is open, we say that the component pieces I and J are *valid*, otherwise they are *invalid*. From now on assume that the component pieces are valid. In this case the shortest path between x and w in P_i is a convex chain (say C_1). Similarly, the shortest path between y and z in P_i is a convex chain (say C_2).

To compute the violation edge bc, where $b \in I$ and $c \in J$, we proceed as follows. We subdivide component pieces I and J into intervals, so that for all points within an interval, the vertex that is tangent on C_1 (and C_2) is the same. This is achieved by scanning C_1 starting from x and drawing lines that are aligned with the edges of C_1. The lines that intersect C_2 are ignored, and the others that intersect wz are retained. (The lines that intersect wz correspond to the consecutive edges of C_1.) Let these lines intersect xy at points $\phi_1, \phi_2, \ldots, \phi_m$, and wz at points $\psi_1, \psi_2, \ldots, \psi_m$, respectively (see Figure 3). We also draw a pair of common tangents of C_1 and C_2. Let these intersect xy at ϕ_0, ϕ_{m+1} and wz at points ψ_0, ψ_{m+1}, respectively. By Lemma 1, the violation edge of the shortest one-violation path that intersects I in the interval $[\phi_i, \phi_{i+1}]$ and J in the interval $[\psi_i, \psi_{i+1}]$, will pass through the same vertex w of P.

For each interval $[\phi_i, \phi_{i+1}]$ and its corresponding pair $[\psi_i, \psi_{i+1}]$ we can compute the length of a shortest one-violation path by optimizing the function that is obtained by adding the lengths of (a) $\Pi_{in}(s, u)$, where u is the parent vertex of I in SPM_s, (b) $\Pi_{in}(t, v)$, where v is the parent vertex of J in SPM_t, (c) the segment bc, where $b \in [\phi_i, \phi_{i+1}]$ and $c \in [\psi_i, \psi_{i+1}]$, (d) the segment ub, and (e) the segment cv. Once we fix the location of $b \in [\phi_i, \phi_{i+1}]$, everything else can be determined in $O(1)$ time from SPM_s and SPM_t. Therefore, for all $b \in [\phi_i, \phi_{i+1}]$, the minimum of $|ub| + |bc| + |cv|$ can be computed in $O(1)$ time. Hence the length of a shortest one-violation path restricted to intervals $b \in [\phi_i, \phi_{i+1}]$ and $c \in [\psi_i, \psi_{i+1}]$ can be computed in $O(1)$ time.

The same method is applied to compute shortest one-violation paths that pass through the vertices of C_2. Finally, the shortest one is considered for a valid pair of component-pieces (I, J). Hence, the length of a shortest one-violation path restricted to I and J can be computed in time proportional to the number of vertices of the pocket P_i. Considering all possible pairs of component-pieces, we can identify the shortest one-violation path from s to t passing through the pocket P_i. By repeating this computation for each pocket, we can compute a shortest one violation path between s and t. We summarize the result in the following theorem.

Theorem 2. *A shortest one-violation path between a pair of points in a simple polygon P consisting of n vertices can be computed in $O(n^3)$ time using $O(n)$ space.*

Proof. A shortest one-violation path either consists of a direct segment between s and t, or a path consisting of two segments, or a path consisting of three or more segments.

We first test whether the segment st is a valid one-violation path. This can be done in $O(n)$ time by scanning the boundary of P and checking the number of intersections between the boundary and st. Now consider the case when a shortest violation path consists of two segments. The turning point of the path cannot be in the exterior of P. Moreover, the turning point needs to be at a vertex of P, otherwise the length of the path can be further improved. For each vertex v of P, we can use the ray shooting data structure to find whether the segment sv and vt intersects the exterior of P at most once. Among all such valid two segment paths, we find the one that has the minimum length. The ray shooting data structure requires $O(n)$ time and for each vertex v we can find whether sv or tv intersects $\overline{P}$ more than once in $O(\log n)$ time.

If a shortest one violation path consists of three or more segments, we adopt Lemma 1. The shortest path maps SPM_s and SPM_t can be computed in $O(n)$ time and space by the algorithms in [11, 13]. We process each pocket P_i separately. Let the number of vertices in P_i be n_i; the number of component-pieces of SPM_s and SPM_t in P_i be μ and ν respectively. We have considered $\mu \times \nu$ pairs of component-pieces (I, J), where $I \in SPM_s$ and $J \in SPM_t$. For each pair (I, J), in $O(n_i)$ time we traverse the entire P_i to form the convex chains C_1 and C_2. If a pair of component-pieces (I, J) is observed to be *valid*, we again traverse C_1 and C_2 in a merge-like fashion to split both I and J into at most n_i intervals in the worst case. As mentioned earlier, since the time of processing each pair of intervals $[\phi_i, \phi_{i+1}] \in I$ and $[\psi_i, \psi_{i+1}] \in J$ needs $O(1)$ time, the processing of a pair of component pieces (I, J) in a pocket P_i needs $O(n_i)$ time. The final result is reported after considering all possible pairs of valid component pieces. The result follows from the fact that the number of pairs of component-pieces (I, J), $I \in SPM_s$ and $J \in SPM_t$ is $O(n^2)$ considering all the pockets of P. □

4 One-violation path problem in a rectilinear polygon

Given a rectilinear polygon P and a pair of points $s, t \in P$, a one-violation rectilinear path is a rectilinear path from s to t such that it has at most one horizontal or one vertical violation edge e that goes outside P, and the intersection of e with $\overline{P}$ is at most one segment. Our objective is to compute a one-violation rectilinear path of minimum length. We will use $\Psi_{in}(s, t)$ to denote a shortest rectilinear path from s to t that stays inside P, and $\Psi_{one}(s, t)$ to denote a shortest one-violation rectilinear path from s to t. We say, a line segment ℓ is "*aligned with*" an edge e of the polygon P if a portion of ℓ overlaps with a portion of e.

Lemma 3. *If s and t are on the boundary of the polygon P, then there exists a minimum length rectilinear path from s to t such that each line segment of this path is* aligned with *some edge of the polygon P.*

Proof. Let Ψ be a minimum length rectilinear path from s to t, and it has a vertical edge e that does not have any portion which is aligned with some edge of the polygon. Here we need to consider two cases depending on whether the next vertical edge e' of Ψ is aligned with some edge of P or not. In the positive case, if we move the edge e horizontally towards e', the length of the path remains unchanged. After being aligned with e', the result holds. In the negative case, the combined edge $e \oplus e'$ starts moving along its next vertical edge keeping the total length of Ψ unchanged. Proceeding similarly, the result holds. Similarly, if a horizontal edge of Ψ is not aligned with a horizontal edge of the polygon, it can be modified in a similar manner so that it aligns with an edge of P. □

If s and t are not on the boundary of P, then there exists a minimum length rectilinear path from s to t whose all the edges are aligned with edges of the polygon P except possibly the edges that are incident on s or t.

Lemma 4. *If there exists a one-violation rectilinear path whose length is less than that of $\Psi_{in}(s, t)$, then we can obtain a minimum length one-violation rectilinear path $\Psi_{one}(s, t)$ whose violation edge is the extension of an edge of P outside the polygon.*

Proof. Let Ψ be a minimum length one-violation rectilinear path whose violation edge $e = [a, b]$ is horizontal, and is not aligned with any edge of P. Its adjacent vertical edges at a and b must be vertically to the opposite direction. Otherwise, we can reduce the path length by moving $[a, b]$ in that (common) direction. Now, we move $[a, b]$ above or below maintaining the same length until it is aligned with the previous or next horizontal edge, say e', of Ψ. If the combined horizontal edge $e \oplus e'$ is not aligned with an edge of the polygon, then it also can be moved in one direction (above or below) maintaining its length unchanged until it becomes aligned with another edge of Ψ or it meets an edge of the polygon. In the first case, the process of moving the merged edge continues, and in the second case, the process stops proving the result. □

As a warm-up, we first explain the method of computing a shortest rectilinear path $\Psi_{in}(s, t)$ in the rectilinear polygon P [22]. We draw a pair of orthogonal line segments $h(s)$ and $v(s)$ (respectively, $h(t)$ and $v(t)$) at the point s (respectively, t). Let the polygon be split in two parts, namely P_{left} and P_{right} on the two sides of $v(s)$. We use $\hat{P}_{left}$ and $\hat{P}_{right}$ to denote the vertices of the polygons P_{left} and P_{right} respectively. Compute the histogram partitioning of P_{left} and P_{right} [22]. Each window of a histogram is the base of the neighboring histogram, and the adjacency relationship among the histograms on each side of $v(s)$ can be represented as a directed tree. We use $\mathcal{T}_{left}$ to denote the 'histogram tree' for the polygon P_{left}. The histogram of a node $v \in \mathcal{T}_{left}$ will be denoted by $H(v)$. In Figure 4, a histogram partitioning and its corresponding tree representation are shown for P_{left}. Next we will consider a method for computing a shortest path from s to t. Let $t \in P_{left}$. We use $H(s)$ and $H(t)$ to denote the histograms containing s and t, respectively; $H(s)$ is the root of the histogram tree for P_{left}. The shortest path from t to s will navigate from $H(t)$ to $H(s)$ in the histogram tree, and the corresponding path will bend at the projection of an edge of the histogram to its base, as shown in Figure 4(a). The histogram partitioning of P and the computation of the histogram tree require $O(n)$ time [16]. The base of each histogram is projected to the base of its parent histogram. These create a set of Steiner points Q_{left}. The bends can take place at the vertices $\hat{P}_{left}$ of P_{left} and at Steiner points in Q_{left}.

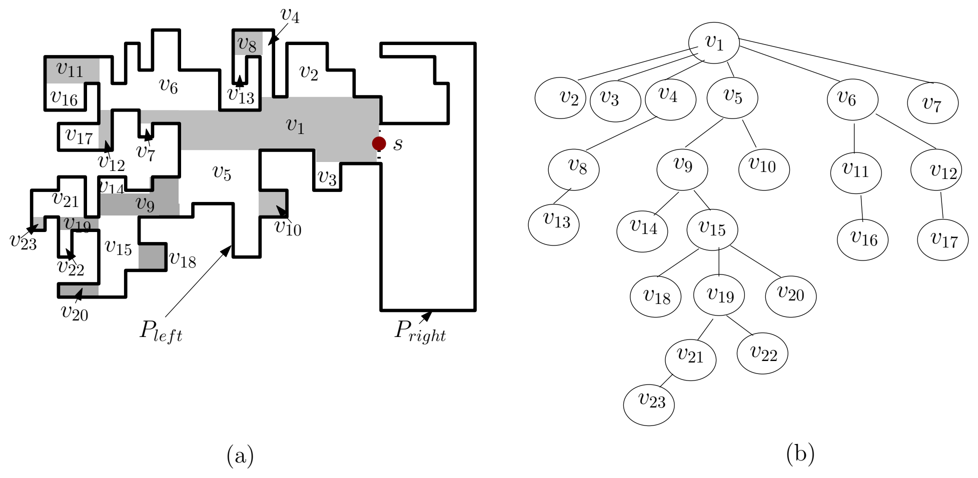


Figure 4: Histogram decomposition of P_{left}, and (b) its histogram tree

Now, if s and t lie in the same histogram, their shortest path is an axis-parallel L-path[7] connecting them. Otherwise, the first bend of the s to t path will be at the projection of t

[7]An L-path consists of a horizontal and a vertical segment incident at a common bend (turn) point

on the base of $H(t)$ [22]. Next time onwards the bends are at the points of $\hat{P}_{left} \cup Q_{left}$. We construct a graph $G = (V, E)$, where $V = \hat{P}_{left} \cup Q_{left} \cup \{s, s_i, i \in \{N, S, E, W\}\} \cup \{t, t_i, i \in \{N, S, E, W\}\}$, where $\{s_i, t_i, i \in \{N, S, E, W\}\}$ are the orthogonal projections of s and t on the boundary of P along the four directions (namely north, south, east and west), respectively. An edge $e \in E$ joins a pair of points $\alpha, \beta \in \hat{P}_{left} \cup Q_{left}$ such that α and β have the same x (or y) coordinates and there is no other point(s) of $\hat{P}_{left} \cup Q_{left}$ in the interval $[\alpha, \beta]$. The shortest path from s to t can be computed by running Dijkstra's algorithm on the graph G.

Observe that $|V| = O(n)$ since each window contributes at most two members to Q and the number of windows in the histogram of P_{left} is equal to the number of nodes in $\mathcal{T}_{left}$, which is at most $|\hat{P}_{left}|$, the number of vertices in P_{left}. The number of edges is also $O(n)$ since each vertex of P is incident on at most 2 edges and each member of Q is incident on at most 3 edges. Thus, Dijkstra's algorithm runs in $O(n \log n)$ time.

We now describe our method for computing a one-violation path of minimum length. By Lemma 4, a violation edge can be the extension $\hat{e}$ of an edge e of P. The extension $\hat{e}$ meets the boundary of P from outside. Note that the extension of e can intersect an edge e' from inside, then it goes outside, and then it intersects another edge e''. Here we need to mention that the violation edge from the boundary of P_{left} may reach a point on the boundary of P_{right}. So, in the graph formulation of computing $\Psi_{one}(s, t)$, we need to handle both P_{left} and P_{right} simultaneously. As in the shortest path problem, we create a graph $G_{one} = (V_{one}, E_{one})$, where $V_{one} = \hat{P} \cup Q \cup R \cup \{s, (s_i, \sigma_i), i \in \{N, S, E, W\}\} \cup \{t, (t_i, \tau_i), i \in \{N, S, E, W\}\}$. Here (i) $\hat{P} = \hat{P}_{left} \cup \hat{P}_{right}$, (ii) Q is the set of vertices generated from the projection of the base of the histograms on their parents' base respectively, as described for the shortest path problem, (iii) R is the orthogonal projections of the vertices in $\hat{P}$ on the boundary of P from outside, (iv) s_i (resp. t_i) is the orthogonal projections of s (resp. t) on the boundary of the polygon from inside in the i-th side and (v) σ_i (resp. τ_i) is the point of intersection of the extended line from s (resp. t) with the boundary of the polygon from outside on the i-th side. The edges $E_{one} = \hat{E} \cup E_{violation}$. The edges in $\hat{E}$ are those defined in the shortest path problem considering the vertices $\hat{P} \cup Q$ and these are tagged as *good*. $E_{violation}$ is the set of all violation edges and these are tagged as *bad*.

For each vertex of P at most two violation edges may exist. Each edge of $E_{violation}$ connects a vertex of $\hat{P}$ and its orthogonal projections ($\in R$) on the boundary of P from outside. The vertices in R and the edges in $E_{violation}$ are generated by sweeping a horizontal (resp. vertical) line over the polygon in $O(n \log n)$ time. Note that, an edge in $E_{violation}$ may intersect many edges in $E_{violation}$. Thus, unlike G, G_{one} may not be a planar graph. However, (i) the number of vertices in R is $O(n)$ since each edge in P is extended in at most two directions, and (ii) the number of edges in $E_{one} = O(n)$ since each vertex of P can define at most two edges of $E_{violation}$. We run Algorithm 1, with number of violations $k = 1$, to compute $\Psi_{one}(s, t)$ and obtain the following result.

Theorem 3. *Given a simple rectilinear polygon P with n vertices and two points s and t inside it, the shortest one-violation rectilinear path from s to t can be computed in $O(n \log n)$ time using $O(n)$ space.*

5 One-stretch violation path problem in a simple polygon

Given a simple polygon P and a pair of points s and t, we will consider the paths from s to t that bends (changes direction) at only the vertices of the polygon. We say a path is a *one stretch violation path* between s and t if it is composed of (a) a path in P from s to a vertex u of P, (b) a path in $\overline{P}$ between u and a vertex v of P, and (c) a path in P between v and t. In order to characterize such paths, consider the convex hull $CH(P)$ of the polygon P, and the associated pockets (see Definition 2).

Definition 3. *The* external stretch *of an* one-stretch violation path *from s to t is a sequence of line segments connecting the vertices of the polygon such that no part of each segment lies in the proper interior of the polygon.*

Observation 3. *A one-stretch violation path $\Pi(s,t)$ from s to t consists of three parts $\Pi_1 \oplus \Pi_2 \oplus \Pi_3$. The edges of Π_1 and Π_3 completely lie inside the polygon, and Π_2 is the external stretch. Note that, any one of Π_1, Π_2 or Π_3 may be empty.*

Observation 4. *If u and v are the two end-vertices of an external-stretch of an* one-stretch violation path*, then it is one of the following two types:*

Type-1: *Both u and v are vertices of the same pocket P_α of P, and the entire external-stretch is inside that pocket (see Figure 5(a)), or*

Type 2: *If they belong to different pockets P_α and P_β, then the external-stretch can be split into three parts $\pi_1 \oplus \pi_2 \oplus \pi_3$, where π_1 is a path from u to a frontier*[8] *of the pocket P_α, π_3 is a path from v to a frontier of the pocket P_β and π_2 connects these two frontiers via the convex hull edges (see Figure 5(b)).*

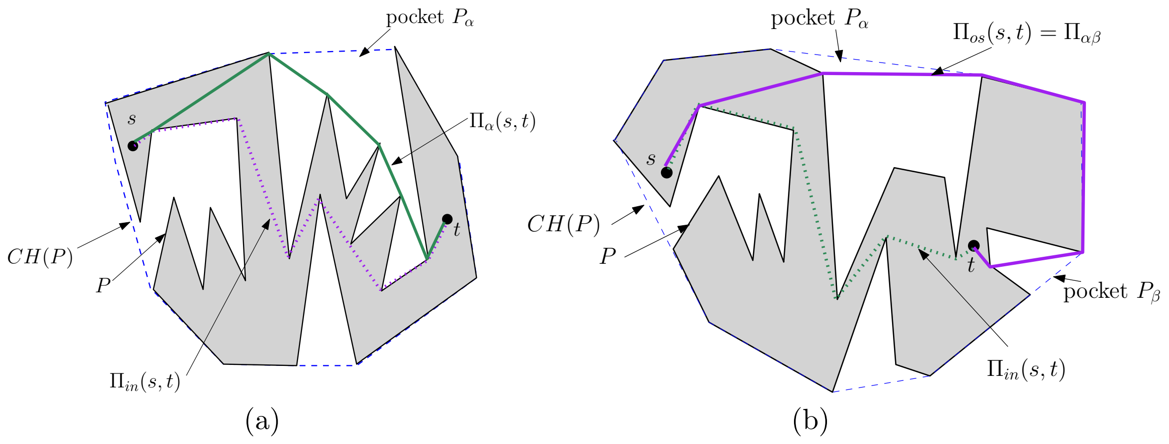


Figure 5: (a) Type-1 $\Pi_{os}(s,t)$, and (b) Type-2 $\Pi_{os}(s,t)$

5.1 Algorithm

We compute the convex hull of the polygon P. Suppose, this generates k pockets, namely $P_1, P_2 \ldots, P_k$. We will use Π_α to denote the minimum length Type-1 path whose stretch

[8]see Definition 2

lies entirely inside the pocket P_α, and $\Pi_{\alpha\beta}$ to denote the minimum length Type-2 path whose external stretch connects two vertices of the pockets P_α and P_β. Thus the length of a shortest one-stretch violation path, denoted by $|\Pi_{os}(s,t)|$, from s to t is given by

$$|\Pi_{os}(s,t)| = \min\{|\Pi_{in}(s,t)|, \min_{\alpha\in\{1,\cdots,k\}} |\Pi_\alpha|, \min_{\alpha,\beta\in\{1,\cdots,k\},\alpha\neq\beta} |\Pi_{\alpha\beta}|\}.$$

We first compute $\Pi_{in}(s,t)$, and the shortest path trees from s and t within P. We use two arrays SP_s and SP_t. For $i = 1,\ldots,n$, the array entry $SP_s[i]$ (respectively, $SP_t[i]$) stores the length of the shortest path of the vertex p_i of P from s (respectively, t). In the following two subsections we explain the methods of computing Π_α's and $\Pi_{\alpha\beta}$'s, respectively.

5.1.1 Type-1 shortest one-stretch violation path

Let us consider the pocket P_α. Let the number of vertices in P_α be m and let the vertices be $\{p_1, p_2, \ldots, p_{m-1}, p_m\}$ in counterclockwise order. Consider a matrix B, whose rows correspond to the vertices $\{p_1, p_2, \ldots, p_{m-1}\}$ of P_α in counterclockwise order and the columns correspond to the vertices $\{p_m, p_{m-1}, \ldots, p_2\}$ in clockwise order. Thus, the rows are numbered as $\{1, 2, \ldots, m-1\}$ from top to bottom and columns are numbered as $\{m, m-1, \ldots, 2\}$ from left to right (see Figure 6(a)). The entries of the matrix B are defined as follows:

$$B[i,j] = \begin{cases} SP_s[i] + SP_t[j] + \chi_{ij} & \text{if } i < j, \text{ and} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Here, χ_{ij} = the length of the shortest path π_{ij} between the vertices $p_i, p_j \in P_\alpha$ inside the pocket P_α. The objective is to find the minimum valued element in this matrix. We now show that the matrix B is a partially monotone reverse rising staircase matrix (see Definition 6). Thus, if we can compute χ_{ij} on demand in $O(f(m))$ time using a data structure that can be computed in $g(m)$ time, then the smallest entry of the matrix B can be computed in $O(g(m) + mf(m)\log\log m)$ time [1]. Similarly, defining the matrix entries as

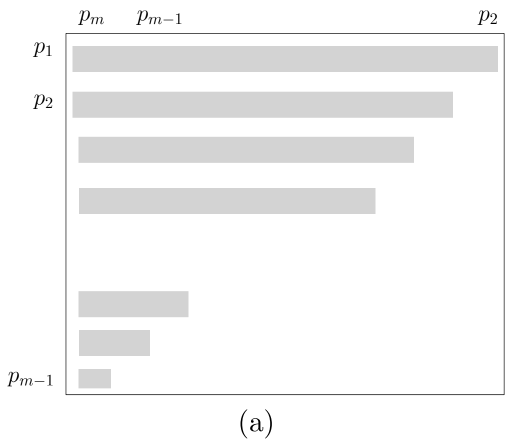
$$B[i,j] = \begin{cases} SP_t[i] + SP_s[j] + \chi_{ij} & \text{if } i < j, \text{ and} \\ \text{undefined} & \text{otherwise,} \end{cases}$$

we execute the same procedure. The minimum of these two values will be the result of processing the pocket P_α. In order to explain the sub-quadratic time processing of the matrix B, we need the following concepts from [1].

Definition 4. *A* 2×2 *matrix* $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ *is said to be* monotone *if* $b < a$ *and* $c < d$ *cannot occur simultaneously.*

Definition 5. *A matrix* B *is said to be* partially monotone *if its every* 2×2 *sub-matrix of* B *that consists of all defined entries, is a monotone matrix.*

Definition 6. *[1] A matrix is said to be a* reverse rising staircase matrix *if (i) the valid entries in each row are consecutive, and (ii) if the valid entries in the* i*-th row span in the column positions from* α_i *to* β_i*, then* $\alpha_i = 1$ *and* β_i*'s are non-increasing for every* $i = 1, 2, \ldots, n$ *(see Figure 6(a)).*



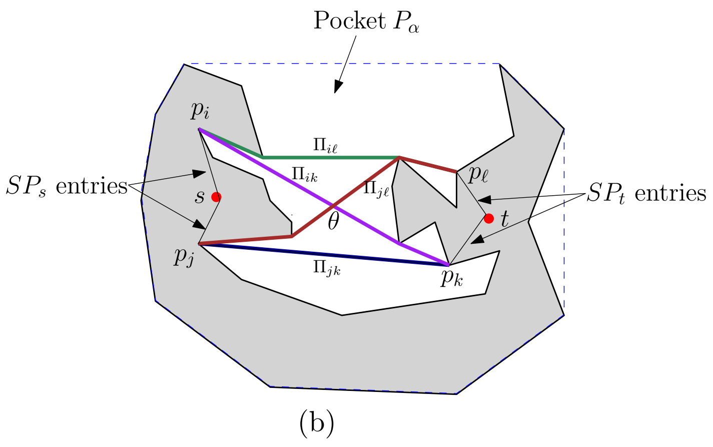


Figure 6: (a) Reverse rising staircase matrix, (b) Illustration of the proof of Lemma 5

Lemma 5. *The matrix B is a reverse rising partial monotone matrix.*

Proof. The structure of the matrix indicates that it is a reverse rising staircase matrix (see Figure 6(a)). In order to prove that it is a partially monotone matrix, we need to show that in every 2×2 sub-matrix of B, if all the entries are defined, then it is a monotone matrix.

For a contradiction, let for a sub-matrix with rows $i < j$ and columns $k > \ell$, $B[i, \ell] > B[i, k]$ and $B[j, k] > B[j, \ell]$ hold simultaneously. Thus, we have

$$SP_s[i] + SP_t[\ell] + \chi_{i\ell} > SP_s[i] + SP_t[k] + \chi_{ik} \dots\dots\dots\dots (1)$$

$$SP_s[j] + SP_t[k] + \chi_{jk} > SP_s[j] + SP_t[\ell] + \chi_{j\ell} \dots\dots\dots\dots (2)$$

In other words, we have

$$SP_t[\ell] + \chi_{i\ell} > SP_t[k] + \chi_{ik} \dots\dots\dots\dots (1')$$

$$SP_t[k] + \chi_{jk} > SP_t[\ell] + \chi_{j\ell} \dots\dots\dots\dots (2')$$

Adding the inequations (1') and (2'), we have

$\chi_{i\ell} + \chi_{jk} > \chi_{ik} + \chi_{j\ell}$.

Due to the configuration of points p_i, p_j, p_k, and p_l along the boundary of P_i, the paths π_{ik} and $\pi_{j\ell}$ must intersect at least at a point, say θ (see Figure 6(b)). The path $\pi_{i\ell}$ may may overlap the path from $p_i \rightsquigarrow \theta$ along π_{ik} and $\theta \rightsquigarrow p_\ell$ along $\pi_{j\ell}$. Similarly, the path π_{jk} may overlap the path segment $p_j \rightsquigarrow \theta$ along $\pi_{j\ell}$ and $\theta \rightsquigarrow p_k$ along π_{ik}. Combining these two, we have $\chi_{ik} + \chi_{j\ell} > \chi_{i\ell} + \chi_{jk}$. Thus, we have a contradiction, and the result follows. □

The pocket P_α can be preprocessed in $g(m) = O(m)$ time for the shortest path queries [10]. This preprocessed data structure along with the arrays SP_s and SP_t enable us to compute $B[i, j]$ in $f(m) = O(\log m)$ time for any i and j for which $B[i, j]$ is defined. Thus, computing Π_α, or in other words, finding the minimum entry in the matrix B requires $O(g(m)+mf(m)\log\log m) = O(m\log m\log\log m)$ time [1]. Since the time for processing the pockets are additive, we have the following result:

Lemma 6. *The total time required for computing a minimum length Type-1 one-stretch violation path is $O(n\log n\log\log n)$.*

5.1.2 Type-2 shortest one-stretch violation path

Lemma 7. *A Type-2 one-stretch violation path $\Pi_{\alpha\beta}, \alpha \neq \beta$, with the two end-points of its external stretch in two different pockets P_α and P_β, respectively, must pass through one of the frontiers of both the pockets.*

Proof. Follows from the fact that the external stretch of $\Pi_{\alpha\beta}$ exits from pocket P_α through one of its frontiers, goes along the boundary of the convex hull, and then enters the pocket P_β through one of its frontiers (see Figure 5(b)). □

As mentioned earlier, the arrays SP_s and SP_t contain the length of the shortest path of every vertex of the polygon P (inside P) from s and t, respectively. We attach two costs $C_s(q)$ and $C_t(q)$ with each vertex q of the convex hull $CH(P)$. These are the minimum cost of connecting q with s and t respectively with a one-stretch violation path through the pocket adjacent to it. If a convex hull vertex q is adjacent to two pockets, then we consider it as two vertices, and two entries are created for this vertex in the array C_s and C_t.

We consider each pocket P_α separately; let q and q' be the frontiers (convex-hull vertices) associated to the pocket P_α. For each vertex $p_i \in P_\alpha$, we compute the length of the shortest path $\delta(p_i, q)$ from p_i to q inside P_α. Next, we compute $C_s(q) = \min_{p_i\in P_\alpha} SP_s[i] + \delta(p_i, q)$, and $C_t(q) = \min_{p_i\in P_\alpha} SP_t[i] + \delta(p_i, q)$. Our next task is to compute $\min_{\alpha\neq\beta} |\Pi_{\alpha\beta}|$. We consider all the hull vertices $\{q_1, q_2, \ldots, q_{2k}\}$ of the k pockets in clockwise order. Observe that,

$$\min_{\alpha\neq\beta} |\Pi_{\alpha\beta}| = \min(\min_{i\neq j} \pi_1(q_i, q_j), \min_{i\neq j} \pi_2(q_i, q_j))$$

where $\pi_1(q_i, q_j) = C_s(q_i) + C_t(q_j)$ + the length of the clockwise path from q_i to q_j along the boundary of $CH(P)$,
$\pi_2(q_i, q_j) = C_s(q_i) + C_t(q_j)$ + the length of the anticlockwise path from q_i to q_j along the boundary of $CH(P)$,

We explain the method of computing $\min_{i\neq j} \pi_1(q_i, q_j)$ using the method of searching for the minimum entry in a partially monotone reverse rising staircase matrix.

We define a $2k \times 2k$ matrix D whose rows correspond to the vertices $\{q_1, q_2, \ldots, q_{2k}\}$ in order, and whose columns correspond to $\{q_{2k}, q_{2k-1}, \ldots q_1\}$ in order. As in the earlier subsection, the entries $D[i, j]$ are undefined if $j \leq i$; otherwise $D[i, j] = \pi_1(q_i, q_j)$.

Lemma 8. *The matrix D is a partially monotone reverse rising staircase matrix.*

Proof. As in the proof of Lemma 5, we prove that for any arbitrary 2×2 sub-matrix $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$, if all the entries are defined, then it is monotone. In other words, if a, b, c, d are all defined then $a > b$ and $c < d$ can not happen simultaneously. Otherwise, the sum of lengths of the clockwise paths $q_i \rightsquigarrow q_k$ and $q_j \rightsquigarrow q_\ell$ should be strictly greater that the sum of lengths of the clockwise paths $q_i \rightsquigarrow q_\ell$ and $q_j \rightsquigarrow q_k$. This is impossible since both the sums are exactly equal. □

Lemma 9. *The time complexity for computing* $\min_{\alpha \neq \beta} |\Pi_{\alpha\beta}|$ *is* $O(n \log \log n)$.

Proof. The processing of a pocket P_α involves computing the shortest path of all its vertices from both of its frontiers. This can be done in $O(m)$ time, where $m = |P_\alpha|$. Since the vertices of each pocket is disjoint from that of other pockets, $C_s(q_i)$ and $C_t(q_i)$ entries of all the hull vertices q_i, $i = 1, 2, \ldots, 2k$ of the polygon P can be computed in $O(n)$ time. Using the C_s and C_t arrays for the hull vertices, each entry of the matrix D can be obtained in $O(1)$ time. Finding the minimum element in the matrix D needs $O(n \log \log n)$ time [1]. Thus, the result follows. □

5.2 Complexity results

Lemmata 6 and 9 lead to the following result.

Theorem 4. *Given a simple polygon P with n vertices and a pair of points $s, t \in P$, the time complexity for computing a minimum length one-stretch violation path from s to t is* $O(n \log n \log \log n)$.

6 Monotone rectilinear path with violations among rectangular obstacles

In this section, we consider the one-violation monotone rectilinear path problems between a pair of points s, t among a set of disjoint axis-parallel rectangular obstacles $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$ inside an axis-parallel rectangle B in $\mathbb{R}^2$. We borrow the following definitions from [2].

Definition 7. [2] **(x-monotone Path)** *A rectilinear path from p to q is said to be x-*monotone *if all horizontal directed edges are from left to right. The vertical segments in the path may be directed in any direction. Similarly, we can define $(-x)$-monotone, y-monotone, and $(-y)$-monotone paths.*

Definition 8. [2] **(xy-monotone Path)** *A rectilinear path from p to q is said to be xy-*monotone *if all horizontal directed edges are from left to right, and all vertical directed edges are from bottom to top. Similarly we can define $(-x)y$, $x(-y)$, and $(-x)(-y)$-monotone paths.*

Definition 9. *[2]* **(Preferred Path)** *A y-preferred xy-path from p is an xy-monotone path which follows the $+y$ direction whenever possible. If it encounters an obstacle it follows the $+x$-direction until the end of the obstacle is reached. Again it resumes to move in the $+y$ direction. The movement continues until it meets the top boundary of the bounding box B. Such a path is denoted by $\Pi_1^y(p)$, where "1" in the subscript indicates the first quadrant with respect to p (i.e., xy-monotone path), and "y" in the superscript indicates the y-preferred path. Similarly, $\Pi_1^x(p)$, $\Pi_2^y(p)$, $\Pi_2^{-x}(p)$, $\Pi_3^{-x}(p)$, $\Pi_3^{-y}(p)$, $\Pi_4^{-y}(p)$, $\Pi_4^x(p)$ are defined.*

Here by a violation we mean a directed line segment that goes *from right to left* or *from top to bottom*. We consider the following two variations of the problem.

- P1: Computing the shortest x-monotone path from s to t with at most one violation. In other words, all the horizontal edges are directed from left to right except at most one edge that is directed from right to left. There is no restriction on the direction of the vertical line segments.

- P2: Computing the shortest xy-monotone path from s to t with at most two violations. In other words, all the horizontal edges on the path from s to t are directed from left to right except at most one edge that is directed from right to left, and all the vertical edges are directed from bottom to top except at most one edge that is directed from top to bottom.

For a given pair of points s and t as the source and target of the desired path, we use the notation S_1 to denote the region bounded by the two staircase paths $\Pi_1^x(s)$ and $\Pi_1^y(s)$. Similarly, the region S_2 is defined by $\Pi_2^y(s)$ and $\Pi_2^{-x}(s)$; the region S_3 is defined by $\Pi_3^{-x}(t)$ and $\Pi_3^{-y}(t)$; the region S_4 is defined by $\Pi_4^{-y}(t)$ and $\Pi_4^x(t)$. The computation of each of these staircase paths and regions need sorting of the members of $\mathcal{R}$ with respect to their bottom or top boundaries depending on the respective cases, and it needs $O(n \log n)$ time in the worst case.

6.1 Problem P1: one-violation rectilinear path

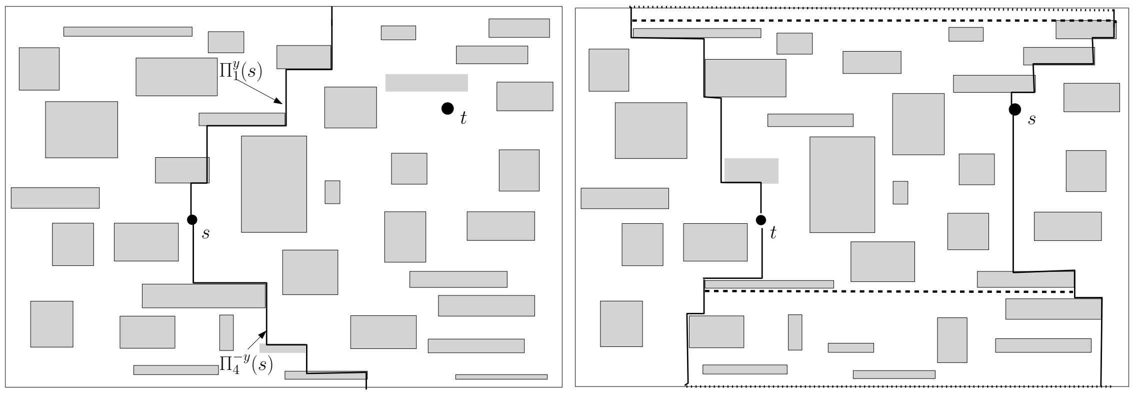
Lemma 10. *For any set $\mathcal{R}$ of disjoint axis parallel rectangular obstacles and any pair of points s, t in $\mathbb{R}^2 \setminus \mathcal{R}$, there always exists a x-monotone path from s to t with at most one violation.*

Proof. Let B be an axis-parallel rectangle that contains all the rectangles in $\mathcal{R}$. We compute $\Pi_1^y(s)$ and $\Pi_2^y(t)$ up to the top boundary of B. Let these meet the top boundary of B at points α and β, respectively. Now, if α is to the left of β then the path $s \rightsquigarrow \alpha \rightarrow \beta \rightsquigarrow t$ has no violation edge; otherwise $\alpha \rightarrow \beta$ is the only violation edge. Note that, this path may be self-intersecting. □

6.1.1 Computation of shortest one-violation rectilinear path

Consider the polygonal line $\Pi^y_{14}(s) = \Pi^y_1(s) \oplus \Pi^{-y}_4(s)$. If the point t lies to the right of $\Pi^y_{14}(s)$ (see Figure 7(a)), then the rectilinear shortest path from s to t is x-monotone, and it can be computed in $O(n \log n)$ time (see de Rezende et al. [20]). Thus, we need to consider the case where t lies to the left of the poly-line $\Pi^y_{14}(s)$ (see Figures 7(b)).

Let us consider the poly-line $\Pi^y_{23}(t) = \Pi^y_2(t) \oplus \Pi^{-y}_3(t)$. Now, consider a horizontal line segment ℓ, that sweeps along vertical direction, keeping its two end-points on $\Pi^y_{14}(s)$ and $\Pi^y_{23}(t)$ respectively. At each instance, if ℓ is not intersected by any one of the members in $\mathcal{R}$, then we compute the length of the rectilinear path $s \rightsquigarrow a \rightarrow b \rightsquigarrow t$, where the path segments $s \rightsquigarrow a$ and $b \rightsquigarrow t$ are along $\Pi^y_{14}(s)$, and $\Pi^y_{23}(t)$ respectively. During this computation, we maintain the minimum length path, and the corresponding horizontal line segment connecting $\Pi^y_{14}(s)$ and $\Pi^y_{23}(t)$ in a temporary storage $\ell^* = [a^*, b^*]$. At the end of the sweep, the path $s \rightsquigarrow a^* \rightarrow b^* \rightsquigarrow t$ is reported.



(a) (b)

Figure 7: Computation of shortest one-violation rectilinear path among obstacles

Theorem 5. *The rectilinear shortest one-violation x-monotone path for a pair of points s and t among a set of disjoint rectangular obstacles can be computed in $O(n \log n)$ time in the worst case.*

Proof. If the point t is to the right of $\Pi^y_{14}(s)$, then both the correctness and time complexity results follow from de Rezende et al. [20]. If t is to the left of $\Pi^y_{14}(s)$, then a feasible one-violation path consists of three parts P_1, P_2, P_3, where P_1 is a staircase path from s to a point in $S_1 \cup S_4$, P_2 is a horizontal line segment directed from right to left, and P_3 is a staircase path from a point in $S_2 \cup S_3$ to t. Such a path must intersect the left (staircase) boundary of the region $S_1 \cup S_4$ and the right (staircase) boundary of the region $S_2 \cup S_3$. The correctness follows from the fact that we have considered all such paths, and chosen the one having the minimum length. Note that $\Pi^y_{14}(s)$, $\Pi^y_{23}(t)$, and ℓ^* can be computed in $O(n \log n)$ time by performing a plane sweep using a horizontal sweep line. Thus, the result follows. □

6.2 Problem P2: two-violation rectilinear path

Lemma 11. *For any set $\mathcal{R}$ of axis parallel rectangular obstacles and a pair of points $s, t \in \mathbb{R}^2 \setminus \mathcal{R}$, there always exists an xy-monotone path from s to t with at most one violation in the horizontal direction and at most one violation in the vertical direction.*

Proof. Let B be an axis-parallel rectangular box containing all the members of $\mathcal{R}$, and let o be its top-left corner. We compute a xy-monotone path Ψ_1 from s up to a point b on the top boundary of B in the region S_1, and another $(-x)(-y)$-monotone path Ψ_2 from t up to a point c on the left boundary of B in the region S_3. If $x(s) < x(t)$ then these two paths may or may not intersect; however if $x(s) > x(t)$ then these paths will never intersect. If Ψ_1 and Ψ_2 intersects at a point a, then the path $s \rightsquigarrow a \rightsquigarrow t$ does not have any violation. Otherwise, the path $s \rightsquigarrow b \rightarrow o \rightarrow c \rightsquigarrow t$ has one horizontal violation edge $b \rightarrow o$ and one vertical violation edge $o \rightarrow c$. $\square$

6.2.1 Computation of shortest two-violation rectilinear path

As mentioned earlier, if the point t lies in the region S_1 bounded by the staircases $\Pi_1^y(s)$ and $\Pi_1^x(s)$, then the shortest path from s to t has no violation edge [20]. Thus, we now concentrate on the case where $t \notin S_1$.

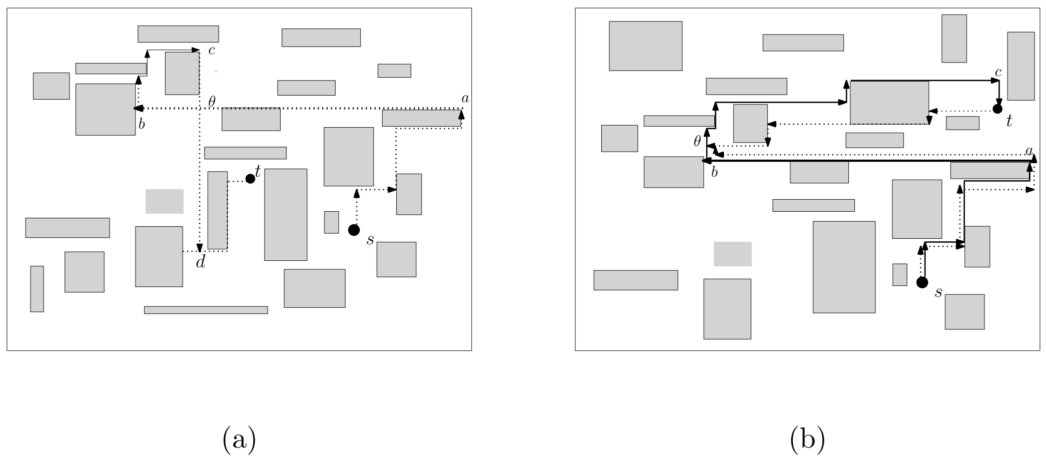


Figure 8: Demonstration of Lemma 12

Lemma 12. *If the shortest two violation xy-monotone path contains exactly two violations (one in both horizontal and vertical directions), then the violation edges are consecutive in the path.*

Proof. On the contrary, let the violation edges are not consecutive, and is of the form: $s \rightsquigarrow a \rightarrow b \rightsquigarrow c \rightarrow d \rightsquigarrow t$, where $s \rightsquigarrow a$ is a xy-monotone path, $a \rightarrow b$ is a horizontal (or vertical) violation edge, $b \rightsquigarrow c$ is a xy-monotone path, and $c \rightarrow d$ is a vertical (or horizontal) violation edge, and $d \rightsquigarrow t$ is a x-monotone path. Here, if the violation edges intersect (at a point, say θ), then we can shorten the path using $s \rightsquigarrow a \rightarrow \theta \rightarrow d \rightsquigarrow t$

(see Figure 8(a)). If the violation edges do not intersect, then consider the regions S_1 and S_3. Note that the first violation edge $a \rightarrow b$ emerges out of the region S_1, and similarly, the second violation edge $c \rightarrow d \rightsquigarrow t$ enters into the region S_3. Observe that, in this case, the path can be shortened and one of the violation edges can be removed by using part of the boundaries $\Pi_3^{-y}(s)$ or $\Pi_3^{-x}(t)$ of the region S_3. (For an illustration, consider Figure 8(b), where such a two violation path $s \rightsquigarrow a \rightarrow b \rightsquigarrow c \rightarrow t$ is shown using solid lines, and the corresponding shortened path $s \rightsquigarrow a \rightarrow b \rightsquigarrow \theta \rightsquigarrow t$ is shown using dotted lines, where $\theta \rightsquigarrow t$ is a xy-monotone path on the boundary of S_3.) □

By Lemma 12, the violation edges occur in pair. We now describe two types of violation paths, where the violations are (a) a horizontal edge followed by a vertical edge (see Figure 9(a)) or (b) a vertical edge followed by a horizontal edge (see Figure 9(b)).

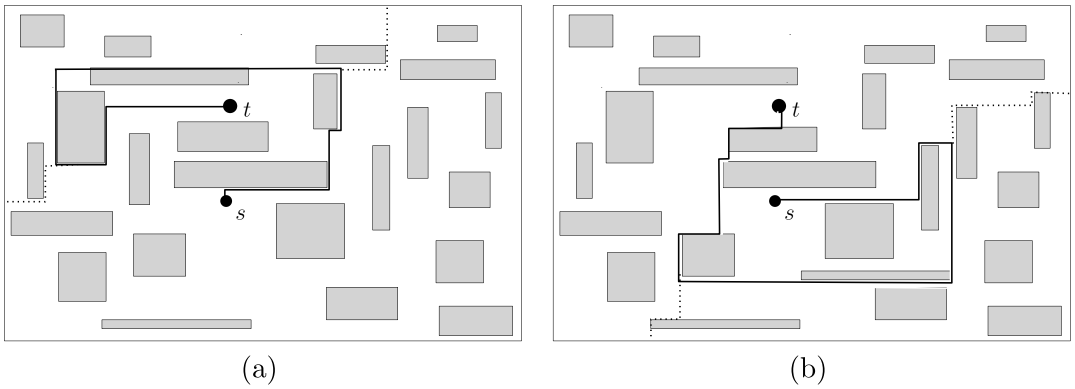


Figure 9: Minimum length two violation path - demonstration of Cases (a) and (b)

We now explain the method of generating the shortest two violation path of type (a). Let Ψ_1 be a xy-monotone path from s in the region S_1, and Ψ_2 be a $(-x)(-y)$-monotone path from t in the region S_3. As mentioned in Lemma 11, any L-path that connects Ψ_1 and Ψ_2 produces a feasible xy-monotone path from s to t with two consecutive violation edges - one in horizontal followed by the other one in the vertical direction. In order to have the minimum length path among the possible paths of type (i) we will connect the left boundary of S_1 (i.e., $\Psi_1 = \Pi_1^{y}(s)$) and the left boundary of S_3 (i.e. $\Psi_2 = \Pi_3^{-x}(t)$) by an L-path composed of a *horizontal violation edge* and a *vertical violation edge*. Thus, we generate a sequence Φ_1 of horizontal violation edges and a sequence Φ_2 of vertical violation edges (shown by blue horizontal lines and green vertical lines respectively, in Figure 10). Each member of Φ_1 (horizontal violation edge) originates from Ψ_1, not intersected by any member of $\mathcal{R}$, and ends at the right boundary of a member of $\mathcal{R}$ or the left boundary of B. These segments are generated by sweeping a horizontal line upwards from s or t depending on which one is above[9], and their right-end moves along Ψ_1. During the generation of these *horizontal violation edges*, we store only those which form a strictly decreasing sequence with respect to the x-coordinates of their left end-points from bottom to top. The reason is that, if a member of $v \in \Phi_2$ does not intersect a member $h \in \Phi_1$ with its left end-point at $x = \alpha$, then it cannot intersect another member $h' \in \Phi_1$ above

[9]If t is above s, then any one-violation path, if exists, will be of smaller length than any two violation path, and we can get an one-violation path of minimum length (if exists) as in Subsection 6.1.1.

h with left end-point at $x > \alpha$. The generation of Φ_1 needs a height-balanced binary tree for storing the members of $\mathcal{R}$ encountered during the sweep at an instance of time in order of their right boundaries. The generated segments in Φ_1 are stored in the form of a stack. Similarly, another sequence Φ_2 of vertical violation edges are generated, which are attached to Ψ_2 and the y-coordinates of their top end-points are strictly decreasing from left to right. The members of Φ_2 are also stored in the form of a stack. Next, in a linear scan over Φ_1 and Φ_2 we can identify a member Φ_1 with minimum y-coordinate that intersects a member of Φ_2 with maximum x-coordinate. Let $ab \in \Phi_1$ and $bc \in \Phi_2$ be these "sticks". Thus, we have a two violation path $s \rightsquigarrow a \rightarrow b \rightarrow c \rightsquigarrow t$ (see Figure 10, where the L-path corresponding to the two violation edges are shown using bold red lines). The entire process takes $O(n \log n)$ time.

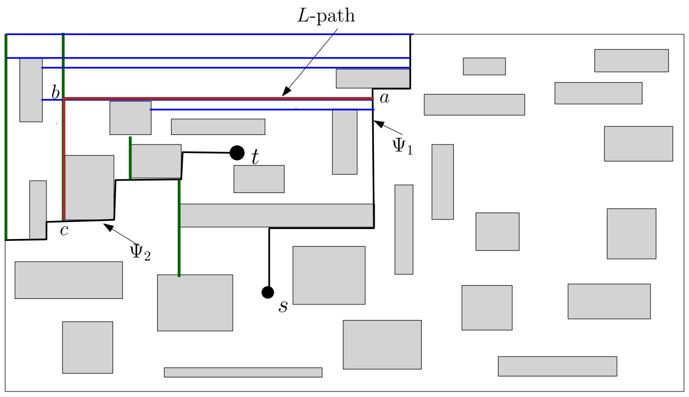


Figure 10: Generation of a two violation path in Case (a)

Similarly, we can compute the shortest path of type (b) by processing $\Psi_1 = \Pi_1^x(s)$ and $\Psi_2 = \Pi_3^{-y}(t)$. Finally, the shortest one among the paths obtained in Case (a) and Case (b) is reported.

Theorem 6. *The minimum length two violation xy-monotone path from s to t among a set of disjoint rectangular obstacles can be computed in $O(n \log n)$ time.*

7 Conclusion

In this paper, we introduced a new concept of shortest path with violations with an aim to reduce the cost of the path. To the best of our knowledge this is the first attempt of studying these variants of the classical geometric shortest path problems. We presented an $O(n^3)$ time algorithm for computing a one-violation shortest path between a pair of points inside a simple polygon. It will be interesting if a sub-cubic time algorithm for this problem can be devised. Another interesting problem in this context is to compute the shortest one-violation path map for point s, where the input is the simple polygon P and the point s, and the output is a data structure which, for a given query point $q \in P$, can report the length of the shortest one-violation path from s to q efficiently.

We also show that, for a given pair of points s and t in rectilinear polygon, the shortest one-violation path from s to t can be reported in $O(n \log n)$ time. We define the one-stretch violation shortest path problem between a pair of points in P, and devised an algorithm that runs in $O(n \log n \log \log n)$ time. Here also one may consider the possibility of having an $O(n \log n)$ time algorithm. Finally, we formulated the one and two violation shortest rectilinear monotone path problems among a set of disjoint rectangular obstacles, and proposed an $O(n \log n)$ time algorithm for both the versions of the problem. For an environment with arbitrary polygonal obstacles, one can easily design a quadratic time algorithm formulating the problem as a k-violation shortest path problem in a graph. Obtaining a sub-quadratic time algorithm, even for the shortest one-violation path problem, in this geometric environment will be interesting.

Acknowledgements

The authors acknowledge Professor Haim Kaplan for discussion on some of the problems presented in this paper. We thank both the referees for making numerous suggestions that led to the improvement of the presentation of the paper.

References

[1] A. Aggarwal and M. Klawe. Applications of generalized matrix searching to geometric algorithms. *Discrete Applied Mathematics*, 27(12):3–23, 1990.

[2] G. Bint, A. Maheshwari, and M. H. M. Smid. xy-monotone path existence queries in a rectilinear environment. In Proc. *CCCG*, pages 35-40, 2012.

[3] J-L. De Carufel, C. Grimm, A. Maheshwari, and M. Smid, Minimizing the Continuous Diameter when Augmenting Paths and Cycles with Shortcuts, 15th Scandinavian Symposium and Workshop on Algorithmic Theory, Reykjavik, Iceland, June 2016.

[4] T. Chan, Low-dimensional linear programming with violations. SIAM Journal on Computing, 34:879–893, 2005.

[5] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6:485-524, 1991.

[6] M. Farshi, P. Giannopoulos, and J. Gudmundsson. Improving the Stretch Factor of a Geometric Network by Edge Augmentation, SIAM Jl. Computing 38(1): 226-240, 2008.

[7] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of ACM*, 34(3):596-615, 1987.

[8] S. K. Ghosh. *Visibility Algorithms in the Plane.* Cambridge University Press, 2007.

[9] U. Große, J. Gudmundsson, C. Knauer, M. Smid, and F. Stehn. Fast Algorithms for Diameter-Optimally Augmenting Paths. 42nd ICALP, LNCS 9134: 678-688, Kyoto, Japan, July 2015.

[10] L. J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygons. *Journal of Computer and System Sciences*, 39:126–152, 1989.

[11] L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-time algorithm for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987.

[12] S. K. Ghosh and D. M. Mount. An output-sensitive algorithm for computing visibility graphs. *SIAM Journal on Computing* 20(5): 888-910, 1991.

[13] J. Hershberger and J. Snoeyink. Computing the minimum length path in a given homotopy class. *Computational Geometry: Theory and Applications*, 4:63-97, 1994.

[14] J. Hershberger and S. Suri. Matrix searching with the shortest-path metric. *SIAM Journal on Computing*, 26(6):1612-1634, 1997.

[15] D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983.

[16] C. Levcopoulos. Fast heuristics for minimum length rectangular partitions of polygons. *Symposium on Computational Geometry*, pages 100–108, 1986.

[17] F. Li and R. Klette. *Euclidean Shortest Paths - Exact or Approximate Algorithms*, Springer, 2011.

[18] J. Matoušek, On geometric optimization with few violated constraints. *Discrete & Computational Geometry*, 14:365–384, 1995.

[19] J. S. B. Mitchell. Geometric shortest paths and network optimization. *Handbook of Computational Geometry*, Elsevier Science Publishers B.V. North-Holland, pages 633–701, 1998.

[20] P. J. de Rezende, D. T. Lee and Y. F. Wu. Rectilinear shortest paths in the presence of rectangular barriers. *Discrete & Computational Geometry*, 4:41–53, 1989.

[21] T. Ross and P. Widemayer, k-violation linear programming. *Information Processing Letters*, 52(2):109–114, 1994.

[22] S. Schuierer. An optimal data structure for shortest rectilinear path queries in a simple rectilinear polygon. *International Journal of Computational Geometry & Applications*, 6(2):205–226, 1996.