Computing Shortest Paths in the Plane with Removable Obstacles*

Pankaj K. Agarwal
Duke University, Durham, NC, USA
pankaj@cs.duke.edu

Neeraj Kumar
University of California, Santa Barbara, CA, USA
neeraj@cs.ucsb.edu

Stavros Sintos
Duke University, Durham, NC, USA
ssintos@cs.duke.edu

Subhash Suri
University of California, Santa Barbara, CA, USA
suri@cs.ucsb.edu

Abstract

We consider the problem of computing a Euclidean shortest path in the presence of *removable* obstacles in the plane. In particular, we have a collection of pairwise-disjoint polygonal obstacles, each of which may be removed at some cost $c_i > 0$. Given a cost budget $C > 0$, and a pair of points s, t, which obstacles should be removed to minimize the path length from s to t in the remaining workspace? We show that this problem is NP-hard even if the obstacles are vertical line segments. Our main result is a fully-polynomial time approximation scheme (FPTAS) for the case of *convex* polygons. Specifically, we compute an $(1+\epsilon)$-approximate shortest path in time $O\left(\frac{nh}{\epsilon^2}\log n \log\frac{n}{\epsilon}\right)$ with removal cost at most $(1+\epsilon)C$, where h is the number of obstacles, n is the total number of obstacle vertices, and $\epsilon \in (0,1)$ is a user-specified parameter. Our approximation scheme also solves a shortest path problem for a *stochastic* model of obstacles, where each obstacle's presence is an independent event with a known probability. Finally, we also present a data structure that can answer s–t path queries in polylogarithmic time, for any pair of points s, t in the plane.

2012 ACM Subject Classification Theory of computation → Shortest paths

Keywords and phrases Euclidean shortest paths, Removable polygonal obstacles, Stochastic shortest paths, L_1 shortest paths

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.5

1 Introduction

We consider a variant of the classical shortest-path problem in the presence of polygonal obstacles, in which the motion planner has the ability to *remove* some of the obstacles to reduce the s–t path length. Formally, let $\mathsf{P} = \{P_1, \ldots, P_h\}$ be a set of h pairwise-disjoint polygonal obstacles in $\mathbb{R}^2$ with n vertices, and let $c_i > 0$ be the cost of removing the obstacle

* Work by Agarwal and Sintos is supported by NSF under grants CCF-15-13816, CCF-15-46392, and IIS-14-08846, by ARO grant W911NF-15-1-0408, and by Grant 2012/229 from the U.S.-Israel Binational Science Foundation. Work by Suri and Kumar is supported by NSF under grant CCF-1525817.

P_i for $i = 1, \ldots, h$. For a path π in $\mathbb{R}^2$, we define its cost, denoted by $c(\pi)$, to be the sum of the costs of obstacles intersecting π, and its length, denoted by $\|\pi\|$, to be its Euclidean length. Given two points $s, t \in \mathbb{R}^2$ and a budget $C > 0$, we wish to compute a path from s to t of minimum length whose cost is at most C.

This *obstacle-removing* shortest path generalizes the classical obstacle-*avoiding* shortest path problem, by giving the planner an option of essentially "tunneling" through obstacles at some cost. Besides an interesting problem in its own right, it is also a natural formulation of tradeoffs in some motion planning settings. For instance, it might be beneficial to remove a few critical blockages in a workspace to significantly shorten an often traveled path, just as an urban commuter may strategically pay money to use certain toll roads or bridges to avoid traffic obstacles. In general, our model with removable obstacles is useful for applications where one can adapt the environment to enable better paths such as urban planning or robot motion planning in a warehouse setting. The problem also generalizes the recent work on obstacle-*violating* paths [18, 25], in which the planner is allowed to enter the forbidden space (obstacles) a fixed *number* of time. For instance, in [25], a shortest s–t path inside a simple polygon is desired, but the path is allowed to travel outside the polygon once. In [18], a shortest path among disjoint convex polygonal obstacles is desired, but is allowed to travel through at most k obstacles. The latter problem is also an obstacle-removing shortest path where *at most k obstacles* can be removed, namely, each obstacle removal has cost 1 and planner's budget is k. We will call this the *cardinality* version of the obstacle-removal to distinguish it from our cost-based model of obstacle removal.

The obstacle removal problem also has a natural connection to path planning under uncertainty. Imagine, for instance, a workspace with n obstacles, the presence of each obstacle is a random event. That is, the presence of the ith obstacle is determined by a Bernoulli trial with (independent) probability β_i. A natural approach to planning a s–t path in such a workspace is to search for a path that is both short and obstacle-free with high probability. Given a desired probability of success β, we can ask: what is the shortest path from s to t that is obstacle free with probability at least β. This problem is easily transformed into our obstacle removal problem where the obstacle probabilities are mapped to obstacle removal cost, and β is mapped to the cost budget C.

Our results. We first show that the obstacle-removing shortest path problem is NP-hard for polygonal obstacles in the plane, even if obstacles are vertical line segments by reducing the well-known Partition problem to it. This is in contrast with the cardinality version of the problem, which can be solved exactly in $O(k^2 n \log n)$ time [18].

Our main result is a fully-polynomial time approximation scheme (FPTAS) when each obstacle is a *convex* polygon. We first define the notion of the *viability* graph G, which is an extension of the well-known visibility graph [11, 13], for geometric paths that can cross obstacles. Using the viability graph, we present a simple algorithm that returns a path with length at most the optimal[1] but cost at most $(1+\epsilon)C$. The approximation algorithm, while simple, has a worst-case time complexity $\Theta(\frac{n^3}{\epsilon}\,\mathrm{polylog}(n))$. Then, we develop a framework for a more efficient and practical approximation algorithm, which also results in a number of related results. Specifically, for any constant $\epsilon > 0$, we can compute a $(1+\epsilon)$-approximate shortest path whose total removal cost is at most $(1+\epsilon)C$ in time $O\left(\frac{nh}{\epsilon^2}\log n \log \frac{n}{\epsilon}\right)$, where h is the number of obstacles and n is the total number of vertices in the obstacles. The main idea behind the improvement is to construct a *sparse* viability graph, with only $O(\frac{n}{\epsilon}\log n)$

[1] The optimal length is always with respect to the budget C.

edges. This approximation scheme immediately gives a corresponding result for the uncertain model of obstacles (see Section 5).

The approximation scheme, as a byproduct, also solves the exact L_1 norm shortest path problem in the *cardinality* model of obstacle removal: that is, in $O(kn \log^2 n)$ time, we can decide which k obstacles to remove for the shortest s–t path, which is roughly a factor of k faster than the L_2-norm result of [18]. Alternatively, we can also decide which k obstacles to remove so that the shortest s–t path has length at most $(1+\epsilon)$ times optimal in $O(\frac{kn}{\epsilon} \log^2 n)$ time. This is again faster than the result from [18] for constant ϵ, if $k = \Omega(\log n)$.

We also construct query data structures for answering approximate obstacle removal shortest path queries. If the source s is fixed (one point queries), we construct a data structure of size $O(\frac{nh}{\epsilon^2} \log n)$ such that, given a query point t, it returns a s–t path of length $(1+\epsilon)$ times the optimal with cost at most $(1+\epsilon)C$ in time $O(\frac{1}{\epsilon} \log^2 n + k_{st})$, where k_{st} is the number of edges in the path. The data structure size can be improved to $O(\frac{n}{\epsilon^2} \log n \log \frac{h}{\epsilon})$ if we only return the length of the path. If both points s, t are given in the query (two point queries), the data structure has size $O(\frac{n^2 h}{\epsilon^3} \log^2 n)$, and the query time is $O(\frac{1}{\epsilon^2} \log^2 n + k_{st})$. The size of the data structure can also be improved to $O(\frac{n^2}{\epsilon^3} \log^2 n \log \frac{h}{\epsilon})$ if we only return the length of the path.

Related work. The problem of computing a shortest path in the presence of polygonal obstacles in the plane is a very well studied problem in computational geometry. See the books [16, 31], survey paper [27], recent papers [5, 8, 9, 10, 20], and references therein for a sample of results. In the classical shortest path problem, obstacles are impenetrable, that is, the shortest path must avoid all the obstacles. Our problem considers a more general scenario where the obstacles can be removed by paying some cost and falls in the broad category of geometric optimization problems where some constraints can be violated [2, 17, 26, 30].

Our problem is also closely related to the problem of computing a shortest path through weighted polygonal regions [23, 24, 28] where the length of a path is defined as the weighted sum of Euclidean or L_1 lengths of the subpaths within each region. However, in our setting there is only a one-time fixed cost for passing through a region, and therefore does not depend on the length of the subpath that lies inside the region.

The stochastic formulation of our problem is also related to some shortest path problems under uncertainty [14, 15, 22, 29]. However, these results assume existence of a graph whose edges have either an existence probability or a distribution over their lengths. In contrast, our definition is purely geometric where the existence of obstacles is an uncertain event. Our problem can also be seen as a variant of geometric bi-criteria shortest path problem [1, 4, 33, 34, 35], as our objective is to compute the shortest path with a constraint on the total cost of obstacles that we remove.

Finally, for most geometric shortest path problems, there are efficient data structures to answer shortest path queries. For instance, the shortest path map [19] has linear size and can answer Euclidean shortest path queries with a fixed source in $O(\log n)$ time. If both s, t are part of the query, quadratic space data structures [7, 21] exist for L_1 shortest path queries and super quadratic space data structures [12] for L_2 shortest path queries. Similar results exist for rectilinear shortest path queries among disjoint weighted rectilinear and convex obstacles [6, 7], and for bi-criteria shortest path problems [4, 33, 35].

Overall, our algorithms entail new techniques because (i) in our problems, paths are allowed to pass through obstacles, (ii) the cost function in our bi-criteria optimization can be quite general and not necessarily a metric.

2 NP-hardness

Consider the decision version of the obstacle-removing shortest-path problem: Given a set P of pairwise-disjoint obstacles along with the cost of each object being removed, two points $s, t \in \mathbb{R}^2$, and two parameters $C, L > 0$, is there a path from s to t of length at most L and cost at most C?

We prove the hardness by a simple reduction from the well-known NP-complete problem Partition. An instance of Partition is a set of n positive integers $A = \{a_1, a_2, \ldots, a_n\}$, and the problem is to decide whether A can be partitioned into two sets A_1 and A_2 such that $W(A_1) = W(A_2) = \frac{1}{2}W(A)$, where $W(S)$ is the sum of the integers in S. We place the source s at $(0, 0)$ and destination t at $(n + 1, 0)$ on the x-axis. We also set $C = \frac{1}{2}W(A)$, $L = \frac{1}{2}W(A) + (n + 1)$ and define a parameter $\delta = \frac{1}{8n}$. For each $i \leq n$ we create a group of obstacles, denoted G_i, which consists of five vertical line segments placed close to each other in the following way. (See also Figure 1.)

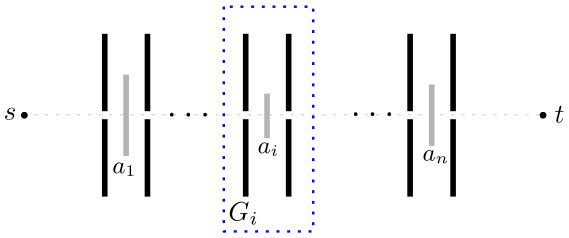

Figure 1 Reduction from Partition. The gray segment in the obstacle group G_i has length a_i and can be crossed by paying a cost a_i. The tall segments are drawn in black and are placed $\pm\delta$ apart from their corresponding gray segment.

- The middle segment e_i^m has length a_i, and has its midpoint on the x-axis. The coordinates of its endpoints are $(i, -a_i/2), (i, a_i/2)$. The cost of this obstacle is a_i.
- At x-coordinates $i - \delta$ and $i + \delta$ we place two vertical segments e_i^l and e_i^r symmetrically along the x-axis – each with point-sized holes on the x-axis and length $2(L + 1)$. The point sized holes split the segment e_i^l (resp. e_i^r) into two disjoint *tall* segments e_i^{lu}, e_i^{ld} (resp. e_i^{ru}, e_i^{rd}), of length $(L + 1)$. Each of these segments has cost $(C + 1)$.

▶ **Lemma 1.** *The set A can be partitioned into two equal-weight subsets if and only if there is a path from s to t of length at most L and cost at most C.*

We thus obtain the following:

▶ **Theorem 2.** *Let P be a set of n disjoint polygonal obstacles in a plane, where each obstacle $P_i \in \mathsf{P}$ has an associated removal cost c_i. Given a source and destination pair of points s, t, a removal budget C and a length L, the problem of deciding if there is a s–t path with cost at most C and length at most L is* NP*-hard.*

3 A Simple $(1 + \epsilon)$-Approximation Algorithm

In this section, we propose a simple polynomial-time approximation scheme for the problem. We begin by noting that an obstacle-removing shortest path only turns at obstacle vertices and crosses the boundary of an obstacle at most twice. While these properties follow easily due to the convexity of P and basic geometry, they are crucial for our algorithms.

The algorithm constructs a *viability graph* $G = (V, E)$, whose nodes are all the obstacle vertices along with s and t. Thus, $|V| = n + 2$. The edges of E correspond to pair of nodes (u, v) for which the line segment uv passes through obstacles of total cost at most C, the cost budget. For each edge $e \in E$, we associate two parameters: cost $c(u, v)$ and length

$\|uv\|$, where $c(u,v)$ is the cost of the segment uv. In the worst-case G has $\Theta(n^2)$ edges. It is important to note that the cost of a path π_{st} in a viability graph is defined as the sum of the costs of its edges, whereas the cost of π_{st} in the plane is defined as sum of costs of all obstacles that it goes through. Moreover, the cost of a path in the plane is at most its cost in the viability graph. If the path crosses each obstacle at most once (which is the case for shortest path among convex obstacles), these two costs are the same.

The following algorithm shows how to compute an approximately optimal path in this viability graph. The main idea is that we construct copies of the vertices and the edges of G to convert the multi-objective problem to a single-objective problem.

Let $\kappa = \min\left(\frac{C}{\min_i c_i}, h\right)$. To simplify the approximation error analysis, we first scale all the costs by κ/C, so that the new target cost is κ. We now construct an auxiliary graph $G' = (V', E')$, with $O\left(\left\lceil \frac{2\kappa|V|}{\epsilon} \right\rceil\right)$ nodes and $O\left(\left\lceil \frac{2\kappa|E|}{\epsilon} \right\rceil\right)$ edges, *whose edges only have the length parameter but not the cost parameter*, as follows. We create $\left\lceil \frac{2\kappa}{\epsilon} \right\rceil + 1$ copies $v_0, v_{\frac{\epsilon}{2}}, v_\epsilon, v_{\frac{3}{2}\epsilon} \ldots, v_\kappa$, for each $v \in V$. Then, for each edge $(u,v) \in E$ with cost c and for each $0 \le i \le \lceil 2\kappa/\epsilon \rceil$, we add the edge $(u_{i\frac{\epsilon}{2}}, v_{j\frac{\epsilon}{2}})$, where $j \le \lceil 2\kappa/\epsilon \rceil$ is the maximum integer with $j\frac{\epsilon}{2} \le i\frac{\epsilon}{2} + c$. All these edge copies have the same length as edge (u,v)—the cost parameter is now implicitly encoded in the edge copies. Finally we add two new vertices s and t in G' and connect them to all s_i and t_i respectively with zero length edges, for $0 \le i \le \lceil 2\kappa/\epsilon \rceil$. We now find the *minimum length* path π from s to t in G', say, using Dijkstra's algorithm, and argue that π is our approximation path.

▶ **Theorem 3.** *Let* P *be a set of h convex obstacles with n vertices, s, t be two obstacle vertices, and $C \in \mathbb{R}$ be a parameter. Let L^* also be the length of the shortest s–t path with cost at most C, and let $G = (V, E)$ be a viability graph induced by this workspace. If there exists a path π^* of length at most αL^* with $\alpha \ge 1$ and cost at most C in the graph G, then a s–t path π with length at most αL^* and cost at most $(1+\epsilon)C$ can be computed in time $O\left(\frac{\kappa}{\epsilon}(|E| + |V| \log \frac{|V|}{\epsilon})\right)$, where $\kappa = \min\left(\frac{C}{\min_i c_i}, h\right)$ and $0 < \epsilon < 1$ is a parameter.*

Proof. First, we construct the auxiliary graph G' as described above. Next, we construct a path π' in G' corresponding to the path π^* in G by mapping edges of π^* to edges in G'. More precisely, let $e = (s, v)$ be the first edge in π^* and let c_e be its cost. Now let $c = 0$ and c' be the value obtained by *rounding down* c_e to the nearest multiple of $\frac{\epsilon}{2}$. We map e to the edge $(s_c, v_{c'})$ in G'. Setting $c = c'$, we repeat the process for all edges in π^*. This gives us the path π' in G' that has the length same as that of π^* (at most αL^*). Clearly, the s–t path π computed using Dijkstra's algorithm on G' must also have length at most αL^*. Moreover, since (scaled) rounded cost of any s–t path in G' is at most κ, the rounded cost of π is also at most κ. Now we only need to bound its original (pre-rounded) cost.

Let C_R be the true (pre-rounded) cost of the path π in the plane and C_A its rounded cost in G'. The approximation error in the cost (due to rounding) is at most $\epsilon/2$ for each obstacle that π passes through, and so if $\bar{k}$ is the number of obstacles π crosses, we have the upper bound $C_R \le C_A + \bar{k}\epsilon/2$. Since $C_A \le \kappa$, we have $C_R \le \kappa + \bar{k}\epsilon/2$. We can bound $\bar{k}$ by considering the following two cases. If $\kappa = C/\min_i c_i$, the minimum cost of an obstacle is 1, and so for each obstacle crossed, the path π incurs a cost of least $1 - \epsilon/2$. Therefore, $\bar{k} \le \frac{\kappa}{1-\epsilon/2}$ and $C_R \le \kappa + \frac{\kappa}{1-\epsilon/2} \cdot \epsilon/2 \le \frac{1}{1-\epsilon/2}\kappa \le (1+\epsilon)\kappa$. Otherwise, we have $\kappa = h$, which trivially implies $\bar{k} \le \kappa$ since h is the total number of obstacles.

In conclusion, we have $C_R \le (1+\epsilon)\kappa$, whose pre-scaled value is $\frac{(1+\epsilon)\kappa}{(\kappa/C)} = (1+\epsilon)C$, as claimed. Finally, the time complexity is dominated by an invocation of Dijkstra's algorithm on the graph G', which has $O(|V|\kappa/\epsilon)$ nodes and $O(|E|\kappa/\epsilon)$ edges. ◀

If G is the viability graph constructed in this section then it always contains the shortest s–t path with cost at most C, i.e. $\alpha = 1$. Hence, by applying Theorem 3 to G we get a path of at most the optimum length and cost at most $(1+\epsilon)C$ in $\Omega(\frac{n^3}{\epsilon})$ time.

In the next section, we show that if we also allow an $(1+\epsilon)$ approximation of the path length, we can improve the running time by roughly an order of magnitude.

4 A Faster $(1+\epsilon)$-Approximation Algorithm

In this section, we describe our algorithm for sparsifying the graph $G = (V, E)$. We augment the graph by adding some vertices so that the number of viability edges can be sharply reduced, while approximately preserving the path lengths within the cost budget. Throughout the following discussion, we will respect the cost budget C, and only allow the path lengths to increase slightly. With that in mind, we use the notation $d_G(u, v)$ to denote the length of the shortest path in G from u to v whose cost is at most C. In this section we only use the definition of the cost of a path with respect to a viability graph. Recall that the cost of a path in a graph is the sum of the costs of the edges in the path.

Our sparse graph $H_\epsilon = (X_\epsilon, T_\epsilon)$ is defined for any $\epsilon > 0$, with $V \subseteq X_\epsilon$, and satisfies the following two conditions:

1. $d_G(u, v) \leq d_{H_\epsilon}(u, v) \leq (1+\epsilon) d_G(u, v)$ for all pairs $u, v \in V$.
2. The number of vertices and edges is $O(\frac{n}{\epsilon} \log n)$, that is, $|X_\epsilon|, |T_\epsilon| = O(\frac{n}{\epsilon} \log n)$.

We construct H_ϵ in two stages. In the first stage we construct a graph $H = (X, \Gamma)$ with $X \supseteq V$, $|X|, |\Gamma| = O(n \log n)$, and $d_G(u, v) \leq d_H(u, v) \leq \sqrt{2} d_G(u, v)$ for all $u, v \in V$. Next, we make $O(1/\epsilon)$ "copies" of H and combine them to construct H_ϵ. Once the graphs H and H_ϵ are constructed, we use the machinery of the previous section, namely Theorem 3, to efficiently find the approximately optimal shortest path within the cost budget.

Recall that all the obstacles in our input are convex, and therefore the shortest path in G does not cross the boundary of an obstacle more than twice. To avoid degenerate cases, we assume that all obstacle vertices are in general position, namely, no three vertices are collinear and all obstacles have non-zero area. We can, therefore, simplify the problem by replacing all the obstacles by their constituent boundary segments, where each obstacle vertex is assigned to its incident segment in the clockwise order. We now allocate the "obstacle removal" cost to these segments as follows: if c_i is the removal cost of obstacle i, then we allocate cost $c_i/2$ to each boundary segment of obstacle i. This ensures that any shortest path crossing the ith obstacle incurs a cost of c_i, while allowing us to reason about the geometry of just line segment obstacles.

We describe the construction of the sparse viability graph by explaining how to sparsify the "neighborhood" of an obstacle vertex, say, p. That is, we show which additional vertices are added and which viability edges are incident to p in the final sparse graph H. To simplify the discussion, we assume that p is at the origin, and we only discuss the edges incident to p that lie in the positive (north-east) quadrant; the remaining three quadrants are processed in the same way.

4.1 An $O(1)$-Approximation Algorithm

In this subsection we describe the construction of $H = (X, \Gamma)$ such that $|X|, |\Gamma| = O(n \log n)$, and $d_G(u, v) \leq d_H(u, v) \leq \sqrt{2} d_G(u, v)$ for all $u, v \in V$.

For a segment pq we use $\|pq\|_1$ to denote its L_1-length, i.e., $\|pq\|_1 = |x_p - x_q| + |y_p - y_q|$, where $p = (x_p, y_p)$ and $q = (x_q, y_q)$. For a polygonal path $\pi = p_0 p_1 \ldots p_k$, we use $\|\pi\|_1$ to

denote its L_1-length, i.e., $\|\pi\|_1 = \sum_{i=1}^{k} \|p_{i-1}p_i\|_1$. We note that $\|\pi\|_1 \leq \sqrt{2}\|\pi\|$. We will construct a graph $H = (X, \Gamma)$ with the following property: For a pair of vertices $u, v \in V$ if G contains a path π from u to v of cost at most C, H contains a path $\bar{\pi}$ from u to v of cost at most C such that $\|\bar{\pi}\|_1 \leq \|\pi\|_1$. Hence $\|\bar{\pi}\| \leq \sqrt{2}\|\pi\|$ and thus $d_H(u, v) \leq \sqrt{2}d_G(u, v)$.

We are now ready to describe the algorithm for constructing H. It is a simple recursive algorithm and consists of the following steps:

1. Let x_m be the median x-coordinate of the points in V. We consider the vertical *split line* $\ell_v : x = x_m$ that partitions V into two almost equal-sized subsets V_l and V_r.
   - **a.** For each point $v \in V$, consider its projection $v' = (x_m, v_y)$ on the line ℓ_v. If $c(v, v') \leq C$, then add the *projection* vertex v' to X and the corresponding edge $e = (v, v')$ to Γ with length $\|vv'\|_1$ and cost $c(v, v')$.
   - **b.** Let s' be the first obstacle segment with positive slope that the projection segment vv' intersects. If s' intersects the split line ℓ_v, we add *bypass* vertices and edges to H as follows. Let v_1 be the point where vv' intersects s', and let v_2 be the point where s' intersects ℓ_v. We add bypass vertices v_1, v_2 on the segment s'. If v_2 lies above v_1, the bypass vertices are considered to be above the segment s', otherwise they are considered below the segment s'. (See also Figure 2.) We add the edges (v, v_1) and (v_1, v_2) to Γ with lengths $\|vv_1\|_1, \|v_1v_2\|_1$ and costs $c(v, v_1), c(v_1, v_2)$, respectively. Observe that $c(v_1, v_2) = 0$.
   - **c.** We repeat the procedure above for the first negative slope segment that vv' intersects.
   - **d.** For two consecutive Steiner vertices w, w' (projection or bypass) on ℓ_v, if $c(w, w') \leq C$, then add the edge $e = (w, w')$ to Γ with length $\|ww'\|_1$ and cost $c(w, w')$.
   - **e.** Recurse on the subsets V_l and V_r until $|V_l|, |V_r| \leq 1$.
2. Repeat the above process but this time using median y-coordinate y_m and a horizontal split line ℓ_h at $y = y_m$.
3. We add edges between consecutive vertices on the boundary of obstacles with cost 0.

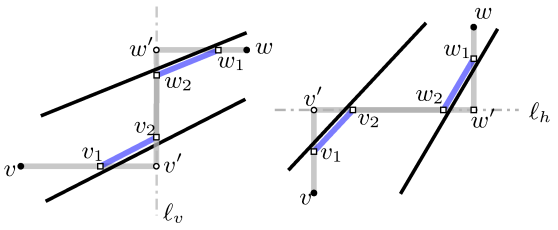

Figure 2 Steiner vertices due to vertical (left) and horizontal (right) split lines. Projections are shown with white dots, bypass vertices as squares, bypass edges shown in blue have cost zero.

At each recursive step of our algorithm, we need to find the first positive (negative) slope obstacle segment intersected by the projection segment vv', and compute the cost of all edges we add. In order to find the first positive (negative) slope segment say s', we can simply perform a point location query in $O(\log n)$ time [32] on positive (negative) slope segments. If s', intersects both the projection segment vv' and the split line passing through v', we add the bypass vertices. For computing the edges costs, observe that bypass edges and the edges on the boundary of obstacles have both cost zero, and all other edges are either horizontal or vertical line segments, so we just need to compute the total cost of obstacle segments intersected by an axis aligned segment. We show how to do this for a horizontal projection segment vv' and all other cases follow similarly. We preprocess all the obstacle segments in a segment tree based data structure S. Using fractional cascading and increasing the fan-out of the segment tree [3,32], a (weighted) counting query runs in $O(\log n)$ time. During each recursive call, we simply query S to compute the cost of the segment vv'. However, we need to be careful in including the cost of the obstacle segment that v lies on. More precisely, if P_i is the obstacle

incident to v, we include the cost $c_i/2$ to the cost of segment vv' only if vv' intersects the interior of P_i (which we can decide in constant time).

We can easily obtain the following lemma.

▶ **Lemma 4.** *Every input vertex adds Steiner vertices on $O(\log n)$ split lines. Moreover, graph H has size $O(n \log n)$ and can be constructed in $O(n \log^2 n)$ time.*

It is important to note here that a similar recursive algorithm was first used by Clarkson et al. [13] to compute L_1 shortest obstacle-avoiding paths in the plane – each vertex was projected on $O(\log n)$ split lines and on the obstacle closest to it in all four directions. This was enough to capture obstacle-avoiding shortest paths (as they lie entirely in free space) but since obstacle-removing shortest paths can also go through obstacles, things get quite complicated. In particular, it is not clear that which of the $O(n)$ nearby obstacles (in each direction) should a vertex be projected on. We address this challenge in Step 1b of our algorithm by adding bypass vertices. Since we need to guarantee that the sparsification preserves the L_1 length as well as the cost of the shortest path, our correctness argument is quite different and can be viewed as a more general form of the result by [13].

4.2 Proof of Correctness

We now prove that $d_H(u, v) \leq \sqrt{2} d_G(u, v)$ for all $u, v \in V$. More precisely, if we set the length of each edge $e = (u, v)$ in G to be $\|uv\|_1$, then we show that $d_H(u, v) \leq d_G(u, v)$. We basically show that for any edge $e = (u, v)$ in G there is a path π_e from u to v in H such that $c(\pi_e) \leq c(u, v)$ and $\|\pi_e\|_1 \leq \|uv\|_1$. This claim is established in Lemma 7, whose proof relies on the following Lemmas 5 and 6.

For convenience, we introduce the notion of the region R_{pq} defined by two obstacle vertices $p, q \in V$. Let $\bar{R}_{pq}$ be the rectangle with p and q as lower left and upper right corners respectively. Now, let s_x (resp. s_y) be the first obstacle segment of positive slope that intersects the two sides of $\bar{R}_{pq}$ below (resp. above) the diagonal pq. We define $R_{pq} = \bar{R}_{pq} \setminus (\mathcal{B}(s_x) \cup \mathcal{A}(s_y))$, where $\mathcal{B}(s_x)$ is the area below segment s_x, and $\mathcal{A}(s_y)$ is the area above s_y. If a segment s_x or s_y does not exist then $\mathcal{B}(s_x) = \emptyset$ and $\mathcal{A}(s_y) = \emptyset$. (See also Figure 3.)

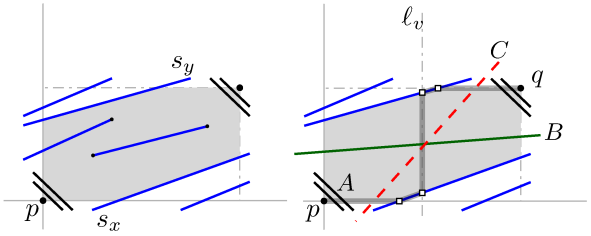

Figure 3 The region R_{pq} is shown shaded. If R_{pq} does not contain obstacle vertices, the type A, B, C obstacle segments that may intersect R_{pq} are shown on the right. Observe that type B and type C segments cannot both exist in R_{pq}.

▶ **Lemma 5.** *Let (p, q) be an edge in G with cost $c(p, q)$. If the region R_{pq} does not contain an obstacle vertex, then there exists a path π_{pq} in H that is entirely contained in R_{pq} such that $\|\pi_{pq}\|_1 = \|pq\|_1$ and $c(\pi_{pq}) = c(p, q)$.*

Proof. Since R_{pq} does not contain any obstacle vertex there are only three types of obstacle segments that intersect R_{pq}. (See also Figure 3.)

1. *Type A* : these obstacle segments have negative slope and intersect both vertical and horizontal segments of R_{pq} adjacent to either p or q.
2. *Type B* : obstacle segments that intersect both vertical segments of R_{pq}.
3. *Type C* : obstacle segments that intersect both horizontal segments of R_{pq}.

It is easy to see that segments of type B and C cannot both exist in R_{pq} because the obstacle segments are non-intersecting. From the construction of H there is always a vertical and a horizontal split line between two obstacle vertices. Let ℓ_v (ℓ_h) be the first vertical (horizontal) split line in the recursion that we consider between the vertices p, q. There are three cases.

- *Only Type A segments exist in R_{pq}.* This case is taken care by the Steiner vertices on the vertical (or horizontal) split line ℓ_v. More precisely, ℓ_v may intersect both s_x and s_y, one of them, or even neither of them. We show what happens in the case where ℓ_v intersects both s_x and s_y and the other cases follow easily. Since there are no obstacle vertices in R_{pq}, s_x, s_y are the first positive slope segments intersected by the projections of p, q on ℓ_v. So we have created bypass vertices p_1, p_2 and q_1, q_2 on s_x, s_y. The path π_{pq} is defined as $\pi_{pq} = pp_1p_2q_2q_1q$ and it is easy to see that $\|\pi_{pq}\|_1 = \|pq\|_1$. Moreover, both π_{pq} and the edge pq cross one time the same set of obstacle segments (only type A), so we have that $c(\pi_{pq}) = c(p, q)$.
- *Type B segments exist in R_{pq}.* In this case, observe that type B edges do not intersect with the horizontal projection segments adjacent to p and q on the vertical split line, and therefore we can use the exact same path π_{pq} as the previous case. The cost of the type B segments needs to be included but since the edge pq must cross these segments, we have that $c(\pi_{pq}) = c(p, q)$.
- *Type C segments exist in R_{pq}.* This case is symmetric to the previous case using the *horizontal* split line ℓ_h. ◀

▶ **Lemma 6.** *Let (p, q) be an edge in G with cost $c(p, q)$. If the region R_{pq} contains one or more obstacle vertices, then there exists an obstacle vertex $r \in R_{pq}$ such that $\|pr\|_1 + \|rq\|_1 = \|pq\|_1$, and $c(p, r) + c(r, q) \leq c(p, q)$.*

Proof. We prove the lemma by exhibiting a vertex r such that (i) the triangle Δprq does not contain any other obstacle vertex, and (ii) no obstacles segment intersects the interiors of both pr and rq. Such a choice of r suffices for our proof since $r \in R_{pq}$ implies that $\|pr\|_1 + \|rq\|_1 = \|pq\|_1$ and we get $c(p, r) + c(r, q) \leq c(p, q)$ because any obstacle segment crossing either pr or rq must also cross pq, otherwise that obstacle segment would terminate inside the triangle which contradicts the choice of r. Next, we show how to find such a vertex.

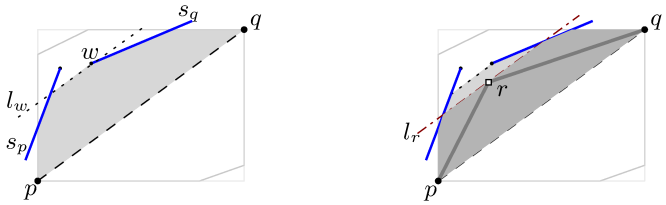

Figure 4 The region T_{pq} is shown shaded on left. If $r \in T_{pq}$ is the vertex closest to pq, then the region $T'_{pq} \subseteq T_{pq}$(shown shaded in dark on right) cannot contain an obstacle vertex.

We restrict our search for this vertex r in a convex polygon $T_{pq} \subseteq R_{pq}$ which we construct in the following way. (See also Figure 4.) Observe that the diagonal pq divides the region R_{pq} into two subsets – one above and one below it. We consider the subset R'_{pq} that contains at least one obstacle vertex. Since, R_{pq} contains at least one obstacle vertex, such a subset always exists. Without loss of generality, we can assume that R'_{pq} lies above pq. Now, let S_{pq} be the set of all obstacle segments that intersect a vertical or a horizontal segment of the boundary $\partial R'_{pq}$, and let $s_p, s_q \in S_{pq}$ be the segments that intersect $\partial R'_{pq}$ closest to p and q respectively. From the endpoints of s_p, s_q that lie in R'_{pq}, let w be the endpoint closest to the segment pq. Moreover, let l_w be the line parallel to pq that passes through w. Now we simply

clip off the region of R'_{pq} that lies above l_w. More precisely, this gives us the quadrilateral $R''_{pq} = R'_{pq} \setminus \mathcal{A}(l_w)$, where we use $\mathcal{A}(s)$ for the region above segment s. Finally, we define the convex polygon $T_{pq} = R''_{pq} \setminus (\mathcal{A}(s'_p) \cup \mathcal{A}(s'_q))$, where s'_p, s'_q are the subsegements of s_p, s_q respectively that lie inside the quadrilateral R''_{pq}.

From the set of obstacle vertices that lie *inside or on the boundary* of T_{pq}, we choose the vertex r to be the one that minimizes the area of the triangle Δprq, or equivalently, be the one that has the minimum distance from the segment pq. Observe that the boundary of region T_{pq} contains the obstacle vertex w, so we will always find one such r. It is easy to see that the triangle Δprq is a subset of T_{pq} and does not contain an obstacle vertex or else it would not have the minimum area. It remains to show that there cannot be an obstacle segment that crosses both pr and rq. To this end, let l_r be a line parallel to pq passing through r. Observe that the region $T'_{pq} = T_{pq} \setminus \mathcal{A}(l_r)$, i.e., the region in T_{pq} that lies below l_r, cannot contain an obstacle vertex by the choice of r. So any obstacle segment s_j that crosses both pr and rq must intersect $\partial R'_{pq}$ at either the vertical segment between p and s_p or the horizontal segment between s_q and q which is a contradiction. (See also Figure 4.) ◀

Finally, we prove the main result of this section.

▶ **Lemma 7.** *Let (p, q) be an edge in G with cost $c(p, q)$. There is a path $\pi_{pq} \in H$ such that $\|\pi_{pq}\|_1 = \|pq\|_1$ and $c(\pi_{pq}) \leq c(p, q)$. Moreover, the path π_{pq} lies in the region R_{pq}.*

Proof. We prove this by induction on the number of obstacle vertices in the region R_{pq}. Our base case is when the region R_{pq} does not contain an obstacle vertex. Applying Lemma 5 gives us the desired path π_{pq} in H. For the inductive step, let j be the number of obstacle vertices in the region R_{pq} and assume that the lemma holds for all edges (u, v) such that the region R_{uv} contains $i < j$ obstacle vertices. Using Lemma 6 we find an intermediate vertex r such that $\|pr\|_1 + \|rq\|_1 = \|pq\|_1$ and $c(p, r) + c(r, q) \leq c(p, q)$. This gives us two disjoint subregions $R_{pr} \subset R_{pq}$ and $R_{rq} \subset R_{pq}$ each with at least one less obstacle vertex than the region R_{pq}. By our induction hypothesis, we get the disjoint subpaths π_{pr} from p to r and π_{rq} from r to q in H. We then join these two paths at vertex r to obtain path π_{pq} that lies within the region R_{pq}. Moreover, we have that $\|\pi_{pq}\|_1 = \|\pi_{pr}\|_1 + \|\pi_{rq}\|_1 = \|pr\|_1 + \|rq\|_1 = \|pq\|_1$ and $c(\pi_{pq}) = c(\pi_{pr}) + c(\pi_{rq}) \leq c(p, r) + c(r, q) \leq c(p, q)$. ◀

4.3 An $(1 + \epsilon)$-Approximation Algorithm

We now describe how to use the preceding construction to define our final sparse graph H_ϵ. A direction in $\mathbb{R}^2$ can be represented as a unit vector $\mathbf{u} \in \mathbb{S}^1$. Let $\mathsf{N} \subset \mathbb{S}^1$ be a set of $O(1/\epsilon)$ unit vectors such that the angle between two consecutive points of N is at most ϵ. For each $\mathbf{u} \in \mathsf{N}$, we construct a graph $H^{\mathbf{u}}$ by running the algorithm in Section 4.1 but regarding $\mathbf{u}$ to be the y axis — i.e., by rotating the plane so that $\mathbf{u}$ becomes parallel to the y-axis and measure L_1-distance in the rotated plane. Set $H_\epsilon = \bigcup_{\mathbf{u} \in \mathsf{N}} H^{\mathbf{u}}$. Notice that the number of vertices and edges in H_ϵ is $O(\frac{n}{\epsilon} \log n)$. The following lemma follows easily by the discussion above.

▶ **Lemma 8.** *For any pair $u, v \in V$, we have that $d_{H_\epsilon}(u, v) \leq (1 + \epsilon) d_G(u, v)$.*

From the above lemma, it follows that the graph H_ϵ preserves pairwise shortest path distances within a factor of $(1 + \epsilon)$ and at most the same cost with graph G. Let L^* be the length of the shortest s–t path in the plane that has cost at most C. Since there exists a s–t path of length at most L^* and cost at most C in the viability graph G, there exists a s–t path in H_ϵ of length $(1 + \epsilon)L^*$ and the same cost. Applying Theorem 3 with $\alpha = (1 + \epsilon)$ on H_ϵ gives the following result.

▶ **Theorem 9.** *Let* P *be a set of* h *convex polygonal obstacles with* n *vertices,* s, t *be two obstacle vertices and* $C \in \mathbb{R}$ *be a parameter. If* L^* *is the length of the shortest* s*–*t *path with cost at most* C*, a* s*–*t *path with length at most* $(1+\epsilon)L^*$ *and cost at most* $(1+\epsilon)C$ *can be computed in* $O(\frac{nh}{\epsilon^2}\log n \log\frac{n}{\epsilon})$ *time.*

5 Shortest Path Queries

We now describe a near-linear space data structure to answer approximate distance queries from a fixed obstacle vertex s subject to the obstacle removal budget in $O(\frac{1}{\epsilon}\log^2 n)$ time. The data structure is then extended to handle two-point shortest path queries in $O(\frac{1}{\epsilon^2}\log^2 n)$ time with near-quadratic space.

The key idea relies on the following observation. Without loss of generality, assume that the points s and t lie in the exterior of all obstacles and let us also assume that s, t were part of the input. Now consider the shortest s–t path in the graph H_ϵ and let t' be the vertex preceding t in this path. It is easy to see that t' must be a Steiner vertex (projection or bypass) as there are no direct edges in H_ϵ between two input vertices that do not lie on the same obstacle. All such edges must cross some split line at Steiner vertices. Therefore, the last edge (t', t) in the path is the segment obtained by projecting t on some split line ℓ. Now, suppose we have precomputed the paths to all Steiner vertices on all split lines, then we can find the shortest path to t by simply finding the neighbor of t' on ℓ. Using Lemma 4, we know that t can be projected on $O(\frac{1}{\epsilon}\log n)$ split lines, which gives $O(\frac{1}{\epsilon}\log n)$ choices for ℓ.

Preprocessing. We apply the algorithm preceding Theorem 3 on the graph H_ϵ that we constructed in the previous section. More precisely, first we multiply the cost of all obstacles by h/C so that the target cost becomes h. Next we create an auxiliary graph H'_ϵ with $O(\frac{h}{\epsilon})$ copies of each vertex in H_ϵ. Running Dijkstra's algorithm on H'_ϵ with source s computes a shortest path to each vertex in H'_ϵ. Now for each vertex v in H_ϵ, we maintain arrays $dist_v, pred_v$ each with size $1 + \frac{h}{\epsilon} = O(\frac{h}{\epsilon})$. We store the length of the shortest path found by Dijkstra's algorithm from s to $v_{i\epsilon}$ (i-th copy of vertex v) at $dist_v(i)$ and its predecessor in $pred_v(i)$. In addition, for each direction $\mathbf{u} \in \mathsf{N}$ that we defined in the previous section we maintain two data structures:

- A segment tree [3] based data structure $S_\mathbf{u}$ that we also used in Section 4.1 to compute the cost of an axis aligned segment in $O(\log n)$ time.
- A balanced search tree $T_\mathbf{u}$ over all the vertical (resp. horizontal) split lines, which is basically the recursion tree corresponding to the algorithm from Section 4.1. More precisely, the root of $T_\mathbf{u}$ is the split line ℓ_m (at the median x-coordinate x_m), and the left and right children are the split lines added during recursive processing of points to the left and right of ℓ_m respectively.

Moreover, for every split line ℓ, we maintain a search tree over all the Steiner vertices that lie on ℓ. Overall, our data structure consists of all arrays $dist_v, pred_v$, $O(\frac{n}{\epsilon})$ search trees, and $O(\frac{1}{\epsilon})$ segment trees $S_\mathbf{u}$. The size of the data structure is $O(\frac{nh}{\epsilon^2}\log n)$ and the preprocessing time is $O(\frac{nh}{\epsilon^2}\log n \log\frac{n}{\epsilon})$.

Query. The query procedure consists of two parts. Given the target query point t, we first find a subset of $O(\frac{1}{\epsilon}\log n)$ split lines L that we need to search. Next, for each line $\ell \in L$, we find the Steiner vertex t' created by projecting t on ℓ and then find the path to t using one of the two neighbors of t' on ℓ. Let v denote a neighbor of t' on ℓ. Finally, we take the shortest of all $O(\frac{1}{\epsilon}\log n)$ candidate paths.

In order to find the subset of split lines we use the search tree $T_{\mathbf{u}}$ over the set of all split lines for a direction $\mathbf{u} \in \mathsf{N}$. For a node $z \in T_{\mathbf{u}}$, if t lies in the region left of split line at z we search the left child, else we search the right child. Searching $T_{\mathbf{u}}$ in this way, we reach a leaf node such that the associated region contains exactly one obstacle vertex w and the query point t. In this case we add a new split line ℓ^* between w and t and add Steiner vertices for the obstacle vertex w on ℓ^*. This gives us a total of $O(\log n) + 1$ split lines per direction that we need to search.

To compute the candidate paths, for a given a split line ℓ, we consider the Steiner vertices – projection t' and bypass t_1, t_2 – for the query point t. The shortest path from ℓ to t may either be $t' \to t$ or $t_2 \to t_1 \to t$. We find a neighbor v of t' or t_2 on ℓ (at most two neighbors are possible). We now consider the section of the path π_{vt} from v to t. If the arrays $dist_v, pred_v$ are not precomputed, which can happen if v is the projection of an obstacle vertex w on the new split line ℓ^*, we set $v = w$ and include the path from w to t along the split line ℓ^* to π_{vt}. (See also Figure 5.)

At this point we have found a vertex v such that $dist_v, pred_v$ are precomputed for all cost values $0, \epsilon, 2\epsilon, \ldots, h$. Since the cost of bypass edges is zero, and all other segments in the path π_{vt} are axis-aligned, we can compute the cost $c(\pi_{vt})$ using the segment tree $S_{\mathbf{u}}$. The remaining cost budget is $h - c(\pi_{vt})$ which we *round up* for lookup in the $dist_v, pred_v$ arrays. More precisely, let j be the smallest integer such that $h - c(\pi_{vt}) \le j\epsilon$, then we compute the length of the candidate s–t path via v as $dist_v(j) + \|\pi_{vt}\|_1$. Finally, we take the minimum over all $O(\frac{1}{\epsilon}\log n)$ choices of v to obtain the shortest path π_{st} using the *pred* arrays. Using a similar argument as in the proof of Theorem 3, one can show that the length of π_{st} is at most $(1+\epsilon)$ times optimal and the cost is $(1+\epsilon)C$. The total query time is $O(\frac{1}{\epsilon}\log n \cdot \log n) = O(\frac{1}{\epsilon}\log^2 n)$.

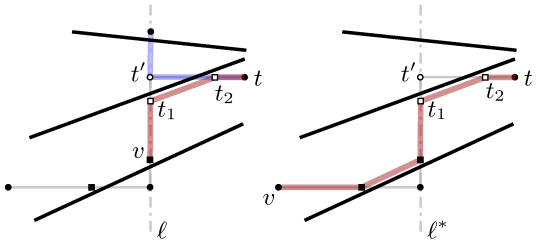

Figure 5 Computing path from a query point t to one of the vertices in H_ϵ – using a split line that already exists in H_ϵ (left) and using a new split line ℓ^* added at query time (right). The suffix path π_{vt} is shown shaded in red.

Instead of computing the path itself, one may ask to just find the length of the shortest s–t path of cost at most C for some query point t. We can answer such queries approximately in $O(\frac{1}{\epsilon}\log^2 n)$ time using $O(\frac{n}{\epsilon^2}\log n \log \frac{h}{\epsilon})$ space. The main idea is that instead of storing $O(\frac{h}{\epsilon})$ distance values in $dist_v$ for cost $0, \epsilon, 2\epsilon, \ldots, \frac{h}{\epsilon}\epsilon$, we store a subset of $O(\frac{1}{\epsilon}\log\frac{h}{\epsilon})$ values. More precisely, we only store the distance values corresponding to the cost $j\epsilon$ where j is the smallest integer such that $\epsilon(1+\epsilon)^i \le j\epsilon$, for all i in $0, 1, 2, \ldots, \log_{1+\epsilon}\frac{h}{\epsilon}$. The size of $dist_v$ arrays for each vertex v is therefore $O(\log_{1+\epsilon}\frac{h}{\epsilon}) = O(\frac{1}{\epsilon}\log\frac{h}{\epsilon})$. Let π_{vt} be the path from v to t. The length of a s–t path via v has length $dist_v(i) + \|\pi_{vt}\|_1$, where i is the smallest integer with $h - c(\pi_{vt}) \le \epsilon(1+\epsilon)^i$. Finally we take the minimum over all $O(\frac{1}{\epsilon}\log n)$ choices of v to obtain the shortest path π_{st}. We can show that the cost of π_{st} is at most $(1+5\epsilon)C$. Constructing the data structure for $\epsilon \leftarrow \epsilon/5$ we obtain the following theorem.

▶ **Theorem 10.** *Let P be a set of h convex polygonal obstacles with n vertices, s be an obstacle vertex, and $C \in \mathbb{R}$ be a parameter. A data structure of $O(\frac{nh}{\epsilon^2}\log n)$ size can be constructed in $O(\frac{nh}{\epsilon^2}\log n \log\frac{n}{\epsilon})$ time such that, given a query point $t \in \mathbb{R}^2$, a path π_{st} can be returned with cost $(1+\epsilon)C$ and length at most $(1+\epsilon)$ times the optimal in $O\big(\frac{1}{\epsilon}\log^2 n + k_{st}\big)$ time, where k_{st} is the number of edges of π_{st}. The length of the path π_{st} can be returned in time $O\big(\frac{1}{\epsilon}\log^2 n\big)$ using a data structure of size $O(\frac{n}{\epsilon^2}\log n \log\frac{h}{\epsilon})$.*

Two point queries. Now we briefly explain how to extend the above data structure to handle two point queries. That is, both s, t are part of the query. During the preprocessing, we store distance values $dist_{uv}$ (similarly $pred_{uv}$) for every pair of vertices u, v in H_ϵ for all cost values $0, \epsilon, 2\epsilon, \ldots, h$. The idea now is to find the neighbor u of s on some split line ℓ_s and neighbor v of t on split line ℓ_t. We compute the cost of paths π_{sv} and π_{vt} as before and set the length of this candidate s–t path to be $dist_{uv}(j) + \|\pi_{su}\|_1 + \|\pi_{vt}\|_1$. Here j is the smallest integer such that $h - c(\pi_{su}) - c(\pi_{vt}) \leq j\epsilon$. We take the minimum across $O(\frac{1}{\epsilon^2}\log^2 n)$ choices of u and v.

▶ **Theorem 11.** *Let* P *be a set of* h *convex polygonal obstacles with* n *vertices, and* $C \in \mathbb{R}$ *be a parameter. A data structure of* $O(\frac{n^2h}{\epsilon^3}\log^2 n)$ *size can be constructed in* $O(\frac{n^2h}{\epsilon^3}\log^2 n \log\frac{n}{\epsilon})$ *time such that, given two query points* $s, t \in \mathbb{R}^2$*, a path* π_{st} *can be returned with cost at most* $(1+\epsilon)C$ *and length at most* $(1+\epsilon)$ *times the optimal in* $O(\frac{1}{\epsilon^2}\log^2 n + k_{st})$ *time, where* k_{st} *is the number of edges of* π_{st}*. The length of the path* π_{st} *can be returned in* $O(\frac{1}{\epsilon^2}\log^2 n)$ *time using a data structure of size* $O(\frac{n^2}{\epsilon^3}\log^2 n \log\frac{h}{\epsilon})$*.*

6 Stochastic Shortest Path

In this section, we consider a stochastic model of obstacles where the existence of each obstacle $P_i \in \mathsf{P}$ is an independent event with known probability β_i. That is, P_i is part of the input with probability β_i and is not part of the input with probability $1 - \beta_i$. We define the probability of path π_{st} as $\prod_{P_i \in S}(1-\beta_i)$ where $S \subseteq \mathsf{P}$ is the set of obstacles that this path goes through (assuming they did not exist). In such a setting, our goal is to compute the approximate shortest path that has probability more than a given threshold $\beta \in (e^{-1}, 1]$.

Let L_β denote the length of the shortest path from s to t with probability at least β. We convert the multiplicative costs to additive costs by setting $c_i = -\ln(1-\beta_i)$ for each obstacle and setting $C = -\ln\beta$. Using Theorem 9, we find a path π_{st} with length $L(\pi_{st}) \leq (1+\epsilon)L_\beta$ and cost $c(\pi_{st}) \leq (1+\epsilon)C$. It can be shown that π_{st} has probability at least $(1-\epsilon)\beta$.

▶ **Theorem 12.** *Let* P *be a set of* h *convex polygonal obstacles with* n *vertices, where each obstacle* $P_i \in \mathsf{P}$ *exists independently with a probability* β_i*,* s, t *be two obstacle vertices and* $\beta \in (e^{-1}, 1]$ *be a parameter. If* L_β *is the length of the shortest* s–t *path with probability at least* β*, a* s–t *path with length at most* $(1+\epsilon)L_\beta$ *and probability at least* $(1-\epsilon)\beta$ *can be computed in* $O(\frac{nh}{\epsilon^2}\log n \log\frac{n}{\epsilon})$ *time.*

Most likely path. We now consider the following question – given a bound L on the length of the path, what is the s–t path with maximum probability? We need a bound on the path length or else there is always a path of probability 1. To answer this question, we can again take negative logarithms of probabilities to transform into an additive cost model and construct the graph H_ϵ as before. Now instead of applying Theorem 3 on H_ϵ, we construct a new graph H^*_ϵ that is exactly the same as H_ϵ, but with length and cost parameters on edges interchanged. More precisely, for an edge $e \in H_\epsilon$ with length l_e and cost c_e, we have an edge $e^* \in H^*_\epsilon$ with length c_e and cost l_e. Next we apply Theorem 3 on the graph H^*_ϵ with $C = (1+\epsilon)L$, and scale all costs with a parameter $O(\frac{n}{C\epsilon}\log n)$, such that the target cost is scaled to $O(\frac{n}{\epsilon}\log n)$. We choose this value because a shortest path in H_ϵ can have $O(\frac{n}{\epsilon}\log n)$ edges. This gives us the following result.

▶ **Theorem 13.** *Let* P *be a set of* h *convex obstacles with* n *vertices,* s, t *be two obstacle vertices, and* $L \in \mathbb{R}$ *be a parameter. If* β_M *is the maximum probability of a path from* s *to* t *with length at most* L*, a path* π_{st} *with length at most* $(1+\epsilon)L$ *and probability at least* β_M *can be computed in* $O(\frac{n^2}{\epsilon^3}\log^2 n \log\frac{n}{\epsilon})$ *time.*

References

1. E. M. Arkin, J. S. Mitchell, and C. D. Piatko. Bicriteria shortest path problems in the plane. In *Proc. 3rd Canad. Conf. Comput. Geom*, pages 153–156, 1991.
2. T. M. Chan. Low-dimensional linear programming with violations. *SIAM J. Comput.*, 34(4):879–893, 2005.
3. T. M. Chan and Y. Nekrich. Towards an optimal method for dynamic planar point location. In *Proc. 56th Symp. Found. Comp. Science*, pages 390–409. IEEE, 2015.
4. D. Z. Chen, O. Daescu, and K. S. Klenk. On geometric path query problems. *Int. J. Comp. Geom. & Applic.*, 11(06):617–645, 2001.
5. D. Z. Chen, J. Hershberger, and H. Wang. Computing shortest paths amid convex pseudodisks. *SIAM J. Comput.*, 42(3):1158–1184, 2013.
6. D. Z. Chen, R. Inkulu, and H. Wang. Two-point L_1 shortest path queries in the plane. In *Proc. 30th Annual Symp. Comput. Geom.*, page 406. ACM, 2014.
7. D. Z. Chen, K. S. Klenk, and H. T. Tu. Shortest path queries among weighted obstacles in the rectilinear plane. *SIAM J. Comput.*, 29(4):1223–1246, 2000.
8. D. Z. Chen and H. Wang. A nearly optimal algorithm for finding L_1 shortest paths among polygonal obstacles in the plane. In *Proc. 19th Europ. Symp. Alg.*, pages 481–492. Springer, 2011.
9. D. Z. Chen and H. Wang. L_1 shortest path queries among polygonal obstacles in the plane. In *Proc. 30th Int. Symp. Theor. Asp. Comp. Science*, volume 20, 2013.
10. D. Z. Chen and H. Wang. Computing shortest paths among curved obstacles in the plane. *ACM Transactions on Algorithms*, 11(4):26, 2015.
11. D. Z. Chen and H. Wang. A new algorithm for computing visibility graphs of polygonal obstacles in the plane. *J. Comput. Geom.*, 6(1):316–345, 2015.
12. Y. Chiang and J. Mitchell. Two-point Euclidean shortest path queries in the plane. In *Proc. 10th ACM-SIAM Annual Symp. Discrete Algorithms*. SIAM, 1999.
13. K. Clarkson, S. Kapoor, and P. Vaidya. Rectilinear shortest paths through polygonal obstacles in $O(n \log^2 n)$ time. In *Proc. 3rd Annual Symp. Comput. Geom.*, pages 251–257. ACM, 1987.
14. T. Feder, R. Motwani, L. O'Callaghan, C. Olston, and R. Panigrahy. Computing shortest paths with uncertainty. *J. Algorithms*, 62(1):1–18, 2007.
15. Y. Gao. Shortest path problem with uncertain arc lengths. *Computers & Mathematics with Applications*, 62(6):2591–2600, 2011.
16. S. K. Ghosh. *Visibility algorithms in the plane*. Cambridge university press, 2007.
17. S. Har-Peled and V. Koltun. Separability with outliers. In *Proc. Int. Symp. Alg. and Comput.*, pages 28–39. Springer, 2005.
18. J. Hershberger, N. Kumar, and S. Suri. Shortest paths in the plane with obstacle violations. In *Proc. 25th Annual Eur. Symp. on Alg.*, volume 87, pages 49:1–49:14, 2017.
19. J. Hershberger and S. Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM J. Comput.*, 28(6):2215–2256, 1999.
20. R. Inkulu and S. Kapoor. Planar rectilinear shortest path computation using corridors. *J. Comput. Geom.*, 42(9):873–884, 2009.
21. M. Iwai, H. Suzuki, and T. Nishizeki. Shortest path algorithm in the plane with rectilinear polygonal obstacles. In *Proc. SIGAL Workshop*, 1994.
22. P. Kamousi, T. M. Chan, and S. Suri. Stochastic minimum spanning trees in Euclidean spaces. In *Proc. 27th Annual Symp. Comput. Geom.*, pages 65–74. ACM, 2011.
23. D. Lee, C. Yang, and C. Wong. Rectilinear paths among rectilinear obstacles. *Discrete Applied Mathematics*, 70(3):185–215, 1996.
24. D. Lee, C.-D. Yang, and T. Chen. Shortest rectilinear paths among weighted obstacle. *Int. J. Comput. Geom. & Appl.*, 1(02):109–124, 1991.

25 A. Maheshwari, S. C. Nandy, D. Pattanayak, S. Roy, and M. Smid. Geometric path problems with violations. *Algorithmica*, pages 1–24, 2016.

26 J. Matoušek. On geometric optimization with few violated constraints. *Discrete & Computational Geometry*, 14(4):365–384, 1995.

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

28 J. S. B. Mitchell and C. H. Papadimitriou. The weighted region problem: finding shortest paths through a weighted planar subdivision. *J. ACM*, 38(1):18–73, 1991.

29 E. Nikolova, M. Brand, and D. R. Karger. Optimal route planning under uncertainty. In *Proc. 16th Int. Conf. Autom. Plann. and Sched.*, volume 6, pages 131–141, 2006.

30 T. Roos and P. Widmayer. k-violation linear programming. *Inf. Process. Lett.*, 52(2):109–114, 1994.

31 J.-R. Sack and J. Urrutia. *Handbook of computational geometry*. Elsevier, 1999.

32 N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Communic. ACM*, 29(7):669–679, 1986.

33 H. Wang. Bicriteria rectilinear shortest paths among rectilinear obstacles in the plane. In *Proc. 33rd Annual Symp. Comput. Geom.*, pages 60:1–60:16, 2017.

34 C. Yang, D. Lee, and C. Wong. On bends and lengths of rectilinear paths: a graph-theoretic approach. *Int. J. Comput. Geom. & Appl.*, 2(01):61–74, 1992.

35 C. Yang, D. Lee, and C. Wong. Rectilinear path problems among rectilinear obstacles revisited. *SIAM J. Comput.*, 24(3):457–472, 1995.