A Divide-and-Conquer Algorithm for Two-Point L_1 Shortest Path Queries in Polygonal Domains

Haitao Wang
Department of Computer Science, Utah State University, Logan, UT 84322, USA
haitao.wang@usu.edu

Abstract

Let $\mathcal{P}$ be a polygonal domain of h holes and n vertices. We study the problem of constructing a data structure that can compute a shortest path between s and t in $\mathcal{P}$ under the L_1 metric for any two query points s and t. To do so, a standard approach is to first find a set of n_s "gateways" for s and a set of n_t "gateways" for t such that there exist a shortest s-t path containing a gateway of s and a gateway of t, and then compute a shortest s-t path using these gateways. Previous algorithms all take quadratic $O(n_s \cdot n_t)$ time to solve this problem. In this paper, we propose a divide-and-conquer technique that solves the problem in $O(n_s + n_t \log n_s)$ time. As a consequence, we construct a data structure of $O(n+(h^2 \log^3 h/ \log\log h))$ size in $O(n+(h^2 \log^4 h/ \log\log h))$ time such that each query can be answered in $O(\log n)$ time.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Computational geometry

Keywords and phrases shortest paths, two-point queries, L_1 metric, polygonal domains

Digital Object Identifier 10.4230/LIPIcs.SoCG.2019.59

Related Version A full version of this paper is available at `https://arxiv.org/abs/1903.01417`.

1 Introduction

Let $\mathcal{P}$ be a polygonal domain of h holes with a total of n vertices, i.e., there is an outer simple polygon containing h disjoint holes and each hole itself is a simple polygon. If $h = 0$, then $\mathcal{P}$ becomes a simple polygon. For any two points s and t, an L_1 *shortest path* from s to t in $\mathcal{P}$ is a path connecting s and t with the minimum length under the L_1 metric.

We consider the two-point L_1 shortest path query problem: Construct a data structure for $\mathcal{P}$ that can compute an L_1 shortest path in $\mathcal{P}$ for any two query points s and t. To do so, a standard approach is to first find a set of n_s "gateways" for s and a set of n_t "gateways" for t such that there exist a shortest s-t path containing a gateway of s and a gateway of t, and then compute a shortest s-t path using these gateways. Previous algorithms [6,7] all take quadratic $O(n_s \cdot n_t)$ time to solve this problem. In this paper, we propose a divide-and-conquer technique that solves the problem in $O(n_s + n_t \log n_s)$ time.

As a consequence, we construct a data structure of $O(n + (h^2 \log^3 h/ \log\log h))$ size in $O(n + (h^2 \log^4 h/ \log\log h))$ time such that each query can be answered in $O(\log n)$ time[1]. Previously, Chen et al. [7] built a data structure of $O(n^2 \log n)$ size in $O(n^2 \log^2 n)$ time that can answer each query in $O(\log^2 n)$ time. Later Chen et al. [6] achieved $O(\log n)$ time queries by building a data structure of $O(n + h^2 \cdot \log h \cdot 4^{\sqrt{\log h}})$ space in $O(n + h^2 \cdot \log^2 h \cdot 4^{\sqrt{\log h}})$ time. The preprocessing complexities of our result improve the previous work [6] by a super polylogarithmic factor. More importantly, our divide-and-conquer technique may be interesting in its own right.

[1] Throughout the paper, unless otherwise stated, when we say that the query time of a data structure is $O(T)$, we mean that the shortest path length can be computed in $O(T)$ time and an actual shortest path can be output in additional linear time in the number of edges of the path.

Related Work. Better results exist for certain special cases. If $\mathcal{P}$ is a simple polygon, then a shortest path in $\mathcal{P}$ with minimum Euclidean length is also an L_1 shortest path [20], and thus by using the data structure in [17,19] for the Euclidean metric, one can build a data structure in $O(n)$ time and space that can answer each query in $O(\log n)$ time; recently Bae and Wang [2] proposed a simpler approach that can achieve the same performance. If $\mathcal{P}$ and all holes of it are rectangles with axis-parallel edges, then ElGindy and Mitra [14] constructed a data structure of $O(n^2)$ size in $O(n^2)$ time that supports $O(\log n)$ time queries.

Better results are also known for *one-point queries* in the L_1 metric [8,11,12,22,24,25], i.e., s is fixed in the input and only t is a query point. In particular, Mitchell [24,25] built a data structure of $O(n)$ size in $O(n \log n)$ time that can answer each such query in $O(\log n)$ time. Later Chen and Wang [8] reduced the preprocessing time to $O(n + h \log h)$ if $\mathcal{P}$ is already triangulated (which can be done in $O(n \log n)$ or $O(n + h \log^{1+\epsilon} h)$ time for any $\epsilon > 0$ [3,4]), while the query time is still $O(\log n)$.

The Euclidean counterparts have also been studied. For one-point queries, Hershberger and Suri [21] built a shortest path map of $O(n)$ size with $O(\log n)$ query time and the map can be built in $O(n \log n)$ time and space. For two-point queries, Chiang and Mitchell [10] built a data structure of $O(n^{11})$ size that can support $O(\log n)$ time queries, and they also built a data structure of $O(n + h^5)$ size with $O(h \log n)$ query time. Other results with tradeoff between preprocessing and query time were also proposed in [10]. Also, Chen et al. [5] showed that with $O(n^2)$ space one can answer each two-point query in $O(\min\{|Q_s|, |Q_t|\} \cdot \log n)$ time, where Q_s (resp., Q_t) is the set of vertices of $\mathcal{P}$ visible to s (resp., t). Guo et al. [18] gave a data structure of $O(n^2)$ size that can support $O(h \log n)$ time two-point queries.

Our Techniques. We follow a similar scheme as in [6,7], using a "path-preserving" graph G proposed by Clarkson et al. [11,12] to determine a set $V_g(q)$ of $O(\log n)$ points (called "gateways") for each query point $q \in \{s, t\}$, such that there exists an L_1 shortest s-t path that contains a gateway in $V_g(s)$ and a gateway in $V_g(t)$. To find a shortest s-t path, the main difficulty is to solve the following sub-problem. Let $\pi(p, q)$ denote a shortest path between two points p and q in $\mathcal{P}$, and let $d(p, q)$ denote its length. Suppose that the gateways of s (resp., t) are formed as a cycle around s (resp., t) such that there is a shortest s-t path containing a gateway of s and a gateway of t (e.g., see Fig. 1). The point s is visible to each gateway p in $V_g(s)$, and thus $d(s, p)$ can be obtained in $O(1)$ time. The same applies to t. Also suppose in the preprocessing we have computed $d(p, q)$ for any $p \in V_g(s)$ and any $q \in V_g(t)$. The goal of the problem is to find $p \in V_g(s)$ and $q \in V_g(t)$ such that the value $d(s, p) + d(p, q) + d(q, t)$ is minimized, so that a shortest s-t path contains both p and q.

To solve the sub-problem, a straightforward method is to try all pairs of p and q with $p \in V_g(s)$ and $q \in V_g(t)$, which is the approach used in both algorithms in [6,7]. This takes $O(n_s \cdot n_t)$ time, where $n_s = |V_g(s)|$ and $n_t = |V_g(t)|$. In [7], both n_s and n_t are $O(\log n)$, which results in $O(\log^2 n)$ query time. In [6], both n_s and n_t are reduced to $O(\sqrt{\log n})$, and thus the query time becomes $O(\log n)$, by using a larger "enhanced graph" G_E (than the original graph G). More specifically, the size of G is $O(n \log n)$ while the size of G_E is $O(n\sqrt{\log n} 2^{\sqrt{\log n}})$ (which is further reduced to $O(h\sqrt{\log h} 2^{\sqrt{\log h}})$ by other techniques [6]).

Our main contribution is to develop an $O(n_s + n_t \log n_s)$ time algorithm for solving the above sub-problem. To this end, we explore the geometric structures of the problem and propose a divide-and-conquer technique, which can be roughly described as follows. For simplicity, suppose we only consider one piece of the gateway cycle of s (e.g., those in the first quadrant of s) and order the gateways of s on that piece by $p_1, p_2, \ldots, p_k$ (e.g., see Fig. 2). Then, in a straightforward way, for p_1, we find a gateway, denoted by q_1, of t that

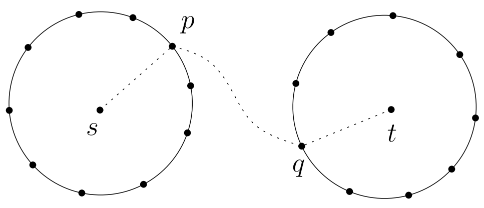

Figure 1 Illustrating the gateways of s and t and a shortest s-t path.

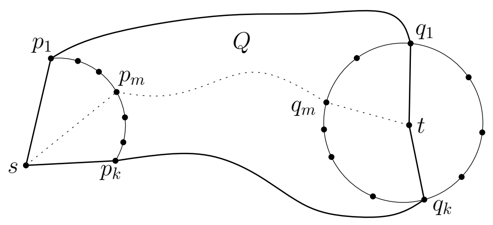

Figure 2 Illustrating our divide-and-conquer scheme.

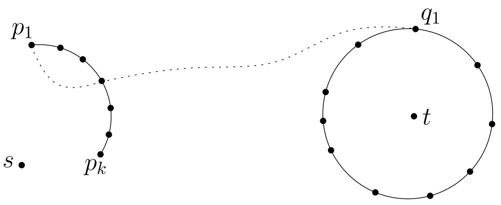

Figure 3 A non-ideal situation: The shortest path from p_1 to q_1 crosses the gateway cycle of s.

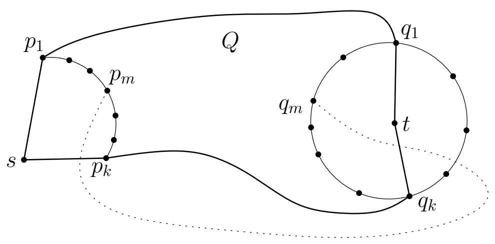

Figure 4 A non-ideal situation: The shortest path from p_m to q_m is not inside the region Q.

minimizes the value $d(p_1, q) + d(q, t)$ for all $q \in V_g(t)$. Similarly, we find such a gateway q_k of t for p_k. Let P_1 be the s-t path $\overline{sp_1} \cup \pi(p_1, q_1) \cup \overline{q_1 t}$. Similarly, let P_2 be the path $\overline{sp_k} \cup \pi(p_k, q_k) \cup \overline{q_k t}$. In the "ideal" situation, the two paths do not intersect except at s and t, and they together form a cycle enclosing a plane region Q that contains all gateways $p_1, p_2, \ldots, p_k$ (e.g., see Fig. 2), and let $V'_g(t)$ be the gateways of t that are also contained in Q. The next step is to process the median gateway p_m of s with $m = \frac{k}{2}$. The key observation is that we only need to consider the gateways in $V'_g(t)$ instead of all the gateways of t, i.e., if a shortest s-t path contains p_m, then there must be a shortest s-t path containing p_m and a gateway in $V'_g(t)$. In this way, we only need to find the point, denoted by q_m, that minimizes the value $d(p_m, q) + d(q, t)$ for all $q \in V'_g(t)$. Further, in the "ideal" situation, the path $P_m = \overline{sp_m} \cup \pi(p_m, q_m) \cup \overline{q_m t}$ is inside the region Q and divides Q into two sub-regions (e.g., see Fig. 2). We then proceed on the two sub-regions recursively.

The above exhibits our algorithm in an "ideal" situation. Our major effort is to deal with the "non-ideal" situations. For example, what if the path P_1 crosses the cycle piece of s (e.g., see Fig. 3), what if the path P_m is not in the region Q (e.g., see Fig. 4), what if $q_1 = q_k$, etc.

Our divide-and-conquer scheme may be somewhat similar to that for two-vertex shortest path queries in planar graphs, e.g., [9,13]. However, a main difference is that in the planar graph case the query vertices are both from the input graph and the gateways are already known for each vertex (more specifically, the gateways in the planar graph case are the "border vertices" of the subgraphs in the decomposition of the input graph by separators), and thus one can compute certain information for the gateways in the preprocessing (many other techniques for shortest path queries in planar graphs, e.g., [15,16,26], also rely on this), while in our problem the gateways are only determined "online" during queries because query points can be anywhere in $\mathcal{P}$. This causes us to develop different techniques to tackle the problem (especially to resolve the non-ideal situations).

It might be tempting to see whether the Monge matrix searching techniques [1,23] can be applied to further shave the logarithmic factor. However, due to the non-ideal situations such as those shown in Fig. 3 and Fig. 4, it is not clear to us whether this is possible.

With the above $O(n_s + n_t \log n_s)$ time algorithm, if both n_s and n_t are bounded by $O(\log n)$, we can only obtain an $O(\log n \log\log n)$ time query algorithm. To reduce the time to $O(\log n)$, we borrow some idea from the previous work [6] to construct a larger graph G_1, so that we can guarantee $n_s = O(\log n)$ and $n_t = O(\log n/\log\log n)$, which leads to an $O(\log n)$ time query algorithm. The size of G_1 is only $O(n \log^2 n/\log\log n)$, which is slightly larger than the original $O(n \log n)$-sized graph G [7, 11, 12] and much smaller than the $O(n\sqrt{\log n}2^{\sqrt{\log n}})$-sized enhanced graph G_E in [6]. Further, by the techniques similar to those used in [6], we can reduce the graph size to $O(h \log^2 h/\log\log h)$.

The rest of the paper is organized as follows. In Section 2, we define notation and review some previous work. In Section 3, we solve the sub-problem discussed above. In Section 4, we present our overall result. For ease of exposition, we make a general position assumption that no two vertices of $\mathcal{P}$ including s and t have the same x- or y-coordinate. Unless otherwise stated, "length" refers to L_1 length and "shortest paths" refers to L_1 shortest paths. Due to the space limit, proofs and many details are omitted but can be found in the full paper.

2 Preliminaries

We introduce some notation and concepts, some of which are borrowed from the previous work [6, 7, 11, 12]. Two points p and q are *visible* to each other if the line segment $\overline{pq}$ is in $\mathcal{P}$. For a point p and a vertical line segment l in $\mathcal{P}$, if there is a point $q \in l$ such that $\overline{pq}$ is horizontal and is in $\mathcal{P}$, then we say p is *horizontally visible* to l and call q the *horizontal projection* of p on l. For any point p in the plane, we use $x(p)$ and $y(p)$ to denote its x- and y-coordinates, respectively. For a path π in $\mathcal{P}$, we use $|\pi|$ to denote its length.

For two points p and q in $\mathcal{P}$, we use $\pi(p, q)$ to denote a shortest path from p to q and define $d(p, q) = |\pi(p, q)|$. For a segment $\overline{pq}$, let $|\overline{pq}|$ denote its length. A path in $\mathcal{P}$ is *x-monotone* if its intersection with any vertical line is either empty or connected. The *y-monotone* is defined similarly. If a path is both x-monotone and y-monotone, then it is *xy-monotone*. Note that an xy-monotone path in $\mathcal{P}$ is a shortest path. Also, if there is an xy-monotone path between p and q in $\mathcal{P}$, then $d(p, q) = |\overline{pq}|$ (although p may not be visible to q).

Let $\mathcal{V}$ denote the set of all vertices of $\mathcal{P}$. To differentiate from the vertices/edges in some graphs we define later, we often refer to the vertices/edges of $\mathcal{P}$ as *polygon vertices/edges*. Let $\partial\mathcal{P}$ denote the boundary of $\mathcal{P}$ (including the boundaries of all the holes). For any point $p \in \mathcal{P}$, if we shoot a ray rightwards from p, let p^r denote the first point of $\partial\mathcal{P}$ hit by the ray and call it the *rightward projection* of p on $\partial\mathcal{P}$. Similarly, we can define the leftward, upward, downward projections of p and denote them by p^l, p^u, p^d, respectively.

A "path-preserving" graph G. Clarkson et al. [11] proposed a graph G for computing L_1 shortest paths in $\mathcal{P}$. We sketch G below since our algorithm will use a modified version of it.

To define G, there are two types of *Steiner points*. For each vertex of $\mathcal{P}$, its four projections on $\partial\mathcal{P}$ are *type-1* Steiner points. Hence, there are $O(n)$ Steiner points on $\partial\mathcal{P}$. The *type-2 Steiner* points are defined on *cut-lines*, which can be organized into a binary tree $\mathcal{T}$, called *the cut-line tree*. Each node u of $\mathcal{T}$ corresponds to a set $\mathcal{V}(u)$ of vertices of $\mathcal{P}$ and stores a cut-line $l(u)$ that is a vertical line through the median x-coordinate of all vertices of $\mathcal{V}(u)$. If u is the root, then $\mathcal{V}(u) = \mathcal{V}$. In general, for the left (resp., right) child v of u, $\mathcal{V}(v)$ consists of all vertices of $\mathcal{V}(u)$ to the left (resp., right) of $l(u)$. For each node $u \in \mathcal{T}$ and each vertex p of $\mathcal{V}(u)$, if p is horizontally visible to $l(u)$, then the horizontal projection of p on $l(u)$ is a type-2 Steiner point. Therefore, $l(u)$ has at most $|\mathcal{V}(u)|$ Steiner points. Since the total size $|\mathcal{V}(u)|$ for all u in the same level of $\mathcal{T}$ is $O(n)$ and the height of $\mathcal{T}$ is $O(\log n)$, the total number of type-2 Steiner points is $O(n \log n)$.

The graph G is thus defined as follows. First of all, the vertex set of G consists of all Steiner points (including all polygon vertices). Hence, it has $O(n \log n)$ nodes. For the edges of G, for each vertex p of $\mathcal{P}$, if q is a Steiner point defined by p, then G has an edge $\overline{pq}$. For each polygon edge e of $\mathcal{P}$, e may contain multiple Steiner points, and G has an edge connecting each adjacent pair of them. Further, for each cut-line l and for any two adjacent Steiner points on l, if they are visible to each other, then G has an edge connecting them.

Clearly, G has $O(n \log n)$ nodes and edges. It was shown in [11, 12] that for any two polygon vertices of $\mathcal{P}$, the shortest path between them in the graph G is also a shortest path in $\mathcal{P}$ (and thus the graph "preserves" shortest paths of the polygon vertices of $\mathcal{P}$).

Gateways. To answer shortest path queries, Chen et al. [7] "insert" the two query points s and t into G by connecting them to some "gateways". Intuitively, the gateways would be the vertices of G that connect to s and t respectively if s and t were vertices of $\mathcal{P}$, and thus they control shortest paths from s to t. Specifically, let $V_g(s, G)$ denote the set of gateways for s, which has two subsets $V_g^1(s, G)$ and $V_g^2(s, G)$ of sizes $O(1)$ and $O(\log n)$, respectively. We first define $V_g^1(s, G)$. For each projection point q of s on $\partial\mathcal{P}$, if v_1 and v_2 are the two Steiner points adjacent to q on the edge of $\mathcal{P}$ containing q, then v_1 and v_2 are in $V_g^1(s, G)$. For $V_g^2(s, G)$, it is defined recursively on the cut-line tree $\mathcal{T}$. Let u be the root of $\mathcal{T}$. If s is horizontally visible to the cut-line $l(u)$, then $l(u)$ is called a *projection cut-line* of s and the Steiner point on $l(u)$ immediately above (resp., below) the horizontal projection s' of s on $l(u)$ is a gateway in $V_g^2(s, G)$ if it is visible to s'. Regardless of whether s is horizontally visible to $l(u)$ or not, if s is to the left (resp., right) of $l(u)$, then we proceed to the left (resp., right) child of u until we reach a leaf of $\mathcal{T}$. Clearly, s has $O(\log n)$ projection cut-lines, on a path from the root to a leaf in $\mathcal{T}$. Hence, $|V_g^2(s, G)| = O(\log n)$. Similarly we can define the gateway set $V_g(t, G)$ for t. As will be shown later, for each gateway p of s, $\overline{sp}$ is in $\mathcal{P}$, and thus $d(s, p) = |\overline{sp}|$. The same applies to t. It is known [7] that if there is a shortest s-t path containing a vertex of $\mathcal{P}$, then there must be a shortest s-t path containing a gateway of s and a gateway of t; otherwise, there must exist a shortest s-t path $\pi(s, t)$ that is xy-monotone with the following property: either $\pi(s, t)$ consists of a horizontal segment and a vertical segment, or $\pi(s, t)$ consists of three segments: $\overline{ss'}$, $\overline{s't'}$, and $\overline{t't}$, where s' is a vertical (resp., horizontal) projection of s and t' is the horizontal (resp., vertical) projection of t on the same polygon edge, and we call such a shortest path a *trivial shortest path*.

A straightforward query algorithm. Given s and t, we can compute $d(s, t)$ as follows. First, we check whether there exists a trivial shortest s-t path, which can be done in $O(\log n)$ time by using vertical and horizontal ray-shootings, after $O(n \log n)$ time (or $O(n + h \log^{1+\epsilon} h)$ time for any $\epsilon > 0$ [3]) preprocessing to build the vertical and horizontal decompositions of $\mathcal{P}$. If yes, then we are done. Otherwise, we compute $V_g(s, G)$ and $V_g(t, G)$ in $O(\log n)$ time after certain preprocessing [6, 7]. Suppose we have computed $d(u, v)$ for any two vertices u and v of G in the preprocessing. Then, $d(s, t) = \min_{p \in V_g(s,G), q \in V_g(t,G)}(|\overline{sp}| + d(p, q) + |\overline{qt}|)$ can be computed in $O(\log^2 n)$ time since both $|V_g(s, G)|$ and $|V_g(t, G)|$ are $O(\log n)$.

The main sub-problem. To reduce the query time, since $|V_g^1(s, G)| = O(1)$ and $|V_g^1(t, G)| = O(1)$, the main sub-problem is to determine the value $\min_{p \in V_g^2(s,G), q \in V_g^2(t,G)}(|\overline{sp}| + d(p, q) + |\overline{qt}|)$. This is the sub-problem we discussed in Section 1. Note that the case $p \in V_g^1(s, G)$ and $q \in V_g^2(t, G)$, or the case $p \in V_g^2(s, G)$ and $q \in V_g^1(t, G)$ can be handled in $O(\log n)$ time.

3 Solving the Main Sub-Problem

In this section, we present an $O(n_s + n_t \log n_s)$ time algorithm for our main sub-problem, where $n_s = |V_g^2(s, G)|$ and $n_t = |V_g^2(t, G)|$.

We consider the vertices of G also as the corresponding points in $\mathcal{P}$. Note that although G preserves shortest paths between all polygon vertices of $\mathcal{P}$, it may not preserve shortest paths for all vertices of G, i.e., for two vertices p and q of G, the shortest path from p to q in G may not be a shortest path in $\mathcal{P}$. For this reason, as preprocessing, for each vertex q of G, we compute a shortest path tree $T(q)$ in $\mathcal{P}$ from q to all vertices of G using the algorithm in [24,25], which can be done in $O(n \log^2 n)$ time since G has $O(n \log n)$ vertices. For each vertex p of G, we use $\pi_q(p)$ to denote the path in $T(q)$ from the root q to p, which is a shortest path in $\mathcal{P}$, and we refer to the edge incident to p as the *last edge* of $\pi_q(p)$; we explicitly store $d(p, q)$ and the last edge of $\pi_q(p)$. Note that $\pi_q(p)$, computed by the algorithm [24,25], has the following property [24,25]: all vertices of the path other than p and q are polygon vertices of $\mathcal{P}$. Doing the above for all vertices q of G takes $O(n^2 \log^3 n)$ time and $O(n^2 \log^2 n)$ space.

Given s and t, following the discussion in Section 2, we assume that there are no trivial shortest s-t paths and there is a shortest s-t path containing a gateway in $V_g^2(s, G)$ and a gateway in $V_g^2(t, G)$. To simplify the notation, let $V(s) = V_g^2(s, G)$ and $V(t) = V_g^2(t, G)$.

A gateway of $V(s)$ is called a *via gateway* if there is a shortest s-t path containing it. Our goal is to find a via gateway, after which a shortest s-t path can be computed in additional $O(\log n)$ time by checking each gateway of t. In the following, we present an $O(n_s + n_t \log n_s)$ time algorithm. Without loss of generality, we assume that the first quadrant of s has a via gateway. Below, we will describe our algorithm only on the gateways of $V(s)$ in the first quadrant of s (our algorithm will run on each quadrant of s separately). By slightly abusing the notation, we still use $V(s)$ to denote the gateways of $V(s)$ in the first quadrant of s.

Before describing our algorithm, we introduce some geometric structures, among which the most important ones are a *gateway region* of s and an *extended gateway region* of t. Chen et al. [7] introduced the gateway region for rectilinear polygonal domains and here we extend the concept to the arbitrary polygonal domain case. In particular, our extended gateway region has several new components that are critical to our algorithm, and it may be interesting in its own right. Due to the space limit, we only discuss some of their properties that are needed for describing our algorithm later, and other properties that are used to prove the correctness of our algorithm are omitted.

The Gateway Region $R(s)$. We define a *gateway region* $R(s)$ for s. Let $p_1, p_2, \ldots, p_k$ be the gateways of s ordered from left to right (e.g., see Fig. 5). Note that each p_i is a type-2 Steiner point on a projection cut-line of s. Let $l_1, l_2, \ldots, l_k$ be the projection cut-lines of s that contain these gateways, respectively, and thus they are also sorted from left to right. It is known [6,7] that the y-coordinates of $p_1, p_2, \ldots, p_k$ are in non-increasing order.

Let s_1 be the intersection of l_1 with the horizontal line through p_k (e.g., see Fig. 6). For each p_i with $i \in [2, k]$, project p_i leftwards horizontally onto l_{i-1} at a point p_i' (note that $p_i' = p_{i-1}$ if $y(p_{i-1}) = y(p_i)$). Define $R(s)$ as the region bounded by the line segments connecting the points s_1, p_1, p_2', p_2, $\ldots$, p_k', p_k, and s_1 in this cyclic order.

We use β_s to denote the boundary portion of $R(s)$ from p_1 to p_k that contains all gateways of $V(s)$. We call β_s the *ceiling*, $\overline{s_1 p_2'}$ the *left boundary*, and $\overline{s_1 p_k'}$ the *bottom boundary* of $R(s)$. We refer to the region $R(s)$ excluding the points on β_s as the *interior* of $R(s)$.

▶ **Observation 1.** *$R(s)$ is in $\mathcal{P}$, and the interior of $R(s)$ does not contain any polygon vertex.*

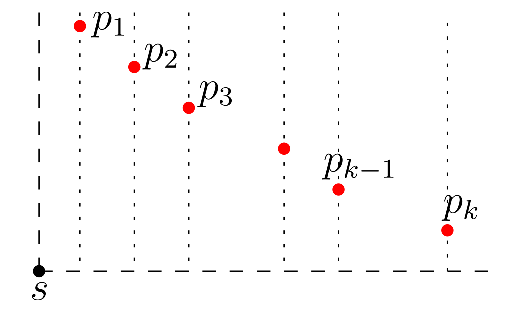

Figure 5 Illustrating the gateways of $V(s)$ and the cut-lines containing them.

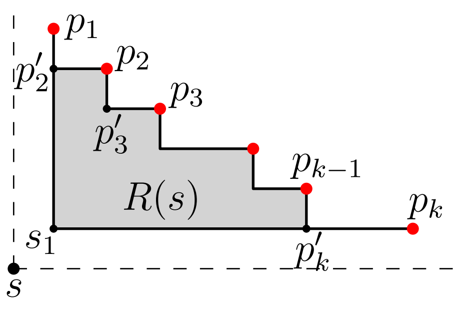

Figure 6 Illustrating the gateway region $R(s)$. The red points are gateways of $V(s)$.

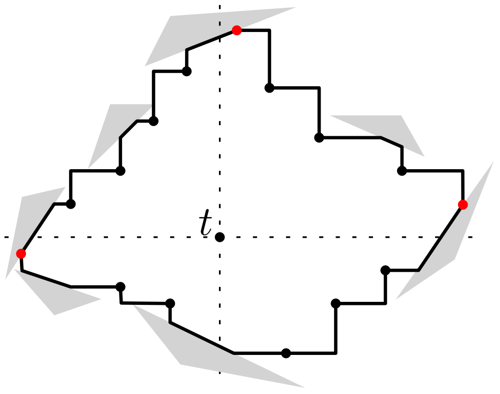

Figure 7 Illustrating $R(t)$, bounded by the solid segments. The black points other than t are all gateways and the three red points are special gateways.

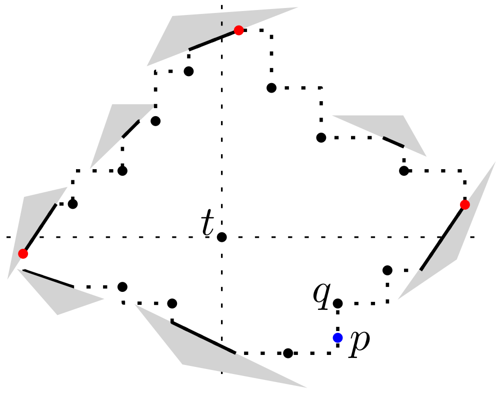

Figure 8 Illustrating the transparent edges (the dotted segments). The three red points are special gateways. For Lemma 1, a point p on a transparent edge as well as an endpoint q of the edge is also shown.

The Extended Gateway Region $R(t)$. For t, we define an *extended gateway region* $R(t)$. Unlike $R(s)$, which does not contain s, $R(t)$ contains t, e.g., see Fig. 7. Instead of giving the detailed definition of $R(t)$, which is quite lengthy, we only discuss several key properties of it.

Let $\mathcal{V}_1$ denote the set consisting of all polygon vertices and their projection points on $\partial\mathcal{P}$.

In general, $R(t)$ is a simple polygon that contains t, with $O(\log n)$ vertices. Let $\partial R(t)$ denote its boundary. Each edge of $\partial R(t)$ is vertical, horizontal, or on a polygon edge. If an edge of $\partial R(t)$ is not on a polygon edge, then we call it a *transparent edge* (e.g., see Fig. 8). It is the transparent edges that separate the interior of $R(t)$ from the outside (i.e., for any point p of $\mathcal{P}$ outside $R(t)$, any path from p to t in $\mathcal{P}$ must intersect a transparent edge of $R(t)$). All gateways of $V(t)$ are on $\partial R(t)$. In addition, at most four points of $\mathcal{V}_1$ are considered as *special gateways* that are also on $\partial R(t)$, and we include them in $V(t)$.

▶ **Lemma 1.** *The point t is visible to each gateway in $V(t)$ and $R(t)$ is in $\mathcal{P}$. For any point p outside $R(t)$, there is a shortest path from p to t that contains a gateway in $V(t)$. For any point p on a transparent edge e, one of the endpoints q of e is a gateway in $V(t)$ such that $\overline{pq} \cup \overline{qt}$ is an xy-monotone (and thus a shortest) path from p to t (e.g., see Fig. 8).*

In addition, the properties of $R(t)$ guarantee that for any point p outside $R(t)$, a shortest path from p to t cannot separate $\partial R(t)$ into two disjoint pieces (see Fig. 9).

If we store the four projections on $\partial\mathcal{P}$ for each Steiner point of G (this costs $O(n \log n)$ additional space), then $R(t)$ can be explicitly computed in $O(\log n)$ time.

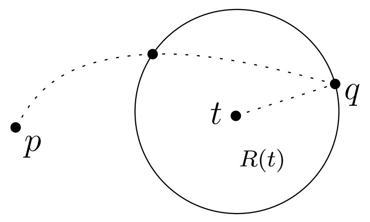

Figure 9 The following situation cannot occur: A shortest path from t to a point p outside $R(t)$ separates the boundary of $R(t)$ (the solid circle) into two or more disjoint pieces.

We remark that $R(s)$ and $R(t)$ are defined differently because s and t are not treated symmetrically in our algorithm. For example, we need $R(t)$ to have the properties in Lemma 1, which are not necessary for $R(s)$. Also, as will be clear later, treating s and t differently helps us to further reduce the complexities of our data structure.

3.1 The Query Algorithm

Consider the gateway region $R(s)$ of s. Note that for any $p_i \in V(s)$, there is always a shortest path from s to p_i containing s_1 as we can show that $\overline{ss_1}$ in $\mathcal{P}$. Recall that we have assumed that there exists a shortest s-t path that contains a gateway of $V(s)$. The above implies that there exists a shortest path from s_1 to t that contains a gateway of $V(s)$, and if we can find such a path, by attaching $\overline{ss_1}$ to the path, we can obtain a shortest s-t path. For convenience, in the following, we will focus on finding a shortest path from s_1 to t that contains a gateway of $V(s)$. By slightly abusing the notation, we still use s to represent s_1. Again, our goal is to compute a via gateway of s in $V(s)$. We first check whether there is a trivial shortest s-t path in $O(\log n)$ time. If yes, we are done. Otherwise, we have the following lemma.

▶ **Lemma 2.** *If $R(t)$ contains a gateway p of $V(s)$, then $\overline{sp} \cup \overline{pt}$ is a shortest s-t path; otherwise, $R(s)$ does not intersect $R(t)$.*

We check whether $R(t)$ contains a gateway of $V(s)$. Due to the properties of $R(t)$, this can be done in $O(n_t + n_s)$ time, as follows. We check the four quadrants of t separately. Let $R_1(t)$ be $R(t)$ in the first quadrant of t. To check whether $R_1(t)$ contains a gateway of $V(s)$, we can simply scan the gateways of $V(s)$ and the gateways of $V(t)$ in $R_1(t)$ simultaneously from left to right (somewhat like merge sort). We do the same for other quadrants of t.

If $R(t)$ contains a gateway of $V(s)$, then by Lemma 2, we have found a shortest s-t path. Otherwise, $R(s)$ and $R(t)$ are disjoint and we proceed as follows. By Lemma 1, for each $p \in V(s)$, $d(p,t) = \min_{q \in V(t)}(d(p,q) + |\overline{qt}|)$, and we call such a gateway q of $V(t)$ minimizing the above value a *coupled gateway* of p and use $c(p)$ to denote it.

Our algorithm will compute a "candidate" coupled gateway $c'(p)$ for every gateway p of $V(s)$ such that if $p \in V(s)$ is via gateway, then $c(p) = c'(p)$. Therefore, once the algorithm is done, the gateway p that minimizes the value $|\overline{sp}| + d(p, c'(p)) + |\overline{c'(p)t}|$ is a via gateway.

For any two points a and b on the ceiling β_s of $R(s)$, we use $\beta_s[a,b]$ to denote the sub-path of β_s between a and b, which is xy-monotone. This means that we can compute $d(p_i, p_j) = |\overline{p_i p_j}|$ in constant time for every two gateways p_i and p_j in $V(s)$.

We consider $V(t)$ as a cyclic list of points in *counterclockwise* order around t (we use "counterclockwise" since the list of $V(s) = \{p_1, p_2, \ldots, p_k\}$ are in clockwise order around s).

We first compute $c(p_1)$ in a straightforward manner, i.e., check every gateway of $V(t)$. This takes $O(n_t)$ time (since $d(p,q)$ for any $p \in V(s)$ and $q \in V(t)$ is already computed in our preprocessing). We also compute $c(p_k)$ in the same way. If there are multiple $c(p_k)$'s, then

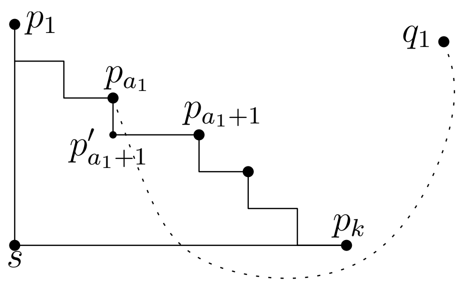

Figure 10 The shortest path $\pi_{q_1}(p_{a_1})$ goes through the interior of $R(s)$.

we let $c(p_k)$ refer to the first one from $c(p_1)$ in the counterclockwise order around t. Further, if there is more than one $c(p_1)$ from the current $c(p_1)$ to $c(p_k)$ in the counterclockwise order, then we update $c(p_1)$ to the one closest to $c(p_k)$. To simplify the notation, let $q_1 = c(p_1)$ and $q_k = c(p_k)$. Note that $q_1 = q_k$ is possible. The following lemma will be useful for circumventing the "non-ideal" situation depicted in Fig. 3. Its correctness relies on the fact that the ceiling β_s of $R(s)$ is xy-monotone (and thus is a shortest path).

▶ **Lemma 3.** *For any p_i of $V(s)$, if $d(p_1, p_i)+d(p_i, q_1) = d(p_1, q_1)$, then $d(p_1, p_j)+d(p_j, q_1) = d(p_1, q_1)$ and $c(p_j) = q_1$ for each $j \in [1, i]$; similarly, if $d(p_k, p_i) + d(p_i, q_k) = d(p_k, q_k)$, then $d(p_k, p_j) + d(p_j, q_k) = d(p_k, q_k)$ and $c(p_j) = q_k$ for each $j \in [i, k]$.*

Let a_1 be the largest index $i \in [1, k]$ such that $d(p_1, p_i) + d(p_i, q_1) = d(p_1, q_1)$, which can be computed in $O(a_1)$ time, as follows. Starting from $i = 2$, we simply check whether $d(p_1, p_i) + d(p_i, q_1) = d(p_1, q_1)$, which can be done in $O(1)$ time since $d(p_1, p_i) = |\overline{p_1 p_i}|$ can be computed in constant time and $d(p_i, q_1)$ has been computed in the preprocessing. If yes, we proceed with $i+1$; otherwise, we stop the algorithm and set $a_1 = i-1$. We call the above a *stair-walking procedure*. The correctness is due to Lemma 3.

Similarly, define b_k to be the smallest index $i \in [1, k]$ such that $d(p_k, p_i) + d(p_i, q_k) = d(p_k, q_k)$. By a symmetric stair-walking procedure, we can compute b_k as well. By Lemma 3, for each $i \in [1, a_1] \cup [b_k, k]$, $c(p_i)$ is known. Hence, if $a_1 \geq b_k$, then $c(p)$ for each $p \in V(s)$ is computed and we can finish the algorithm. Otherwise, we proceed as follows.

Recall that $\pi_{q_1}(p_{a_1})$ is the shortest path from q_1 to p_{a_1} obtained from the shortest path tree $T(q_1)$, and $\pi_{q_k}(p_{b_k})$ is from $T(q_k)$. Lemmas 4 and 5 are for dealing with the non-ideal situation in which $\pi_{q_1}(p_{a_1})$ (resp., $\pi_{q_k}(p_{b_k})$) goes through the interior of $R(s)$ (see Fig. 10).

▶ **Lemma 4.** *(1) $\pi_{q_1}(p_{a_1})$ contains a point in the interior of $R(s)$ only if its last edge (i.e., the edge incident to p_{a_1}) intersects the bottom boundary of $R(s)$. (2) $\pi_{q_k}(p_{b_k})$ contains a point in the interior of $R(s)$ only if its last edge (i.e., the edge incident to p_{b_k}) intersects the left boundary of $R(s)$.*

▶ **Lemma 5.** *If the last edge of $\pi_{q_1}(p_{a_1})$ intersects the bottom boundary of $R(s)$, or the last edge of $\pi_{q_1}(p_{b_k})$ intersects the left boundary of $R(s)$, then p_j for each $j \in [a_1 + 1, b_k - 1]$ cannot be a via gateway.*

Due to our preprocessing, we can check in constant time whether the last edge of $\pi_{q_1}(p_{a_1})$ intersects the bottom boundary of $R(s)$. Similarly, we can check whether the last edge of $\pi_{q_1}(p_{b_k})$ intersects the left boundary of $R(s)$. If the answer is yes for either case, then by Lemma 5, we can stop the algorithm (i.e., no need to compute the coupled gateways for any p_i with $i \in [a_1 + 1, b_k - 1]$). Otherwise, by Lemma 4, neither $\pi_{q_1}(p_{a_1})$ nor $\pi_{q_1}(p_{b_k})$ contains a point in the interior of $R(s)$. We proceed as follows.

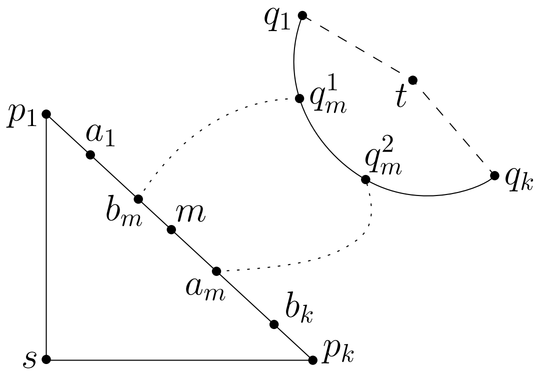

Figure 11 Illustrating a schematic view of the indices: a_1, b_m, m, a_m, and b_k.

Recall that $q_1 = q_k$ is possible. Depending on whether $q_1 = q_k$, there are two cases. In the sequel, we describe our algorithm for the *unequal case* $q_1 \neq q_k$. The *equal-case* $q_1 = q_k$ can be reduced to the unequal case, which is omitted and can be found in the full paper.

In the following, we assume that $q_1 \neq q_k$. Since $q_1 \neq q_k$, q_1 and q_k partition the cyclic list $V(t)$ into two sequential lists, one of which has q_1 as the first point and q_k as the last point following the counterclockwise order around t, and we use $V_t(1,k)$ to denote that list. Lemma 6 follows from our particular way of defining q_1 and q_k.

▶ **Lemma 6.** *The two paths $\pi_{q_1}(p_{a_1})$ and $\pi_{q_k}(p_{b_k})$ do not intersect.*

For any i and j with $1 \leq i \leq j \leq k$, we use the interval $[i,j]$ to represent the gateways $p_i, p_{i+1}, \ldots, p_j$. Our algorithm works on the interval $[1,k]$ and $V_t(1,k)$.

▶ **Lemma 7.** *For any gateway p_j with $j \in [a_1+1, b_k-1]$, if p_j is a via gateway, then it has a coupled gateway in $V_t(1,k)$.*

By Lemma 7, to compute the candidate coupled gateways for all p_i with $i \in [a_1+1, b_k-1]$, we only need to consider $V_t(1,k)$. In the following, we work on the problem recursively. We may consider each recursive step as working on a subproblem, denoted by $([i',j'],[i,j],V_t(i,j))$ with $[i',j'] \subseteq [i,j] \subseteq [1,k]$, where the goal is to find candidate coupled gateways from a sublist $V_t(i,j)$ of $V_t(1,k)$ for the gateways in $[i',j']$, and further, there exist a shortest path from p_{a_i} to the first point of $V_t(i,j)$ and a shortest path from p_{b_j} to the last point of $V_t(i,j)$ such that the two paths do not intersect and neither path contains a point in the interior of $R(s)$. Initially, our subproblem is $([a_1+1, b_k-1],[1,k],V_t(1,k))$. We proceed as follows.

If $b_k - 1 = a_1 + 1$, then the interval $[a_1+1, b_k-1]$ has only one gateway p. We simply check all gateways of $V_t(1,k)$ to find the point q that minimizes the value $d(p,q)+d(q,t)$ among all $q \in V_t(1,k)$, and then return q as the candidate coupled gateway of p. The algorithm can stop. Otherwise, we proceed as follows.

Let $m = \lfloor (a_1+b_k)/2 \rfloor$. We compute a gateway in $V_t(1,k)$ that minimizes the value $d(p_m,q)+d(q,t)$ for all $q \in V_t(1,k)$, and in case of a tie, we use q_m^1 and q_m^2 to refer to the first and the last such gateways in $V_t(1,k)$, respectively. Let $V_t(1,m)$ and $V_t(m,k)$ denote the sublists of $V_t(1,k)$ from q_1 to q_m^1 and from q_m^2 to q_k, respectively.

Define a_m to be the largest index $i \in [m, b_k-1]$ such that $d(p_m,q_m^2) = d(p_m,p_i)+d(p_i,q_m^2)$ and b_m the smallest index $i \in [a_1+1,m]$ such that $d(p_m,q_m^1) = d(p_m,p_i)+d(p_i,q_m^1)$. See Fig. 11. We can compute a_m and b_m by a similar stair-walking procedure as before. With Lemma 7 and similarly as Lemma 3, for each $i \in [b_m, m-1]$, if p_i is a via gateway, then q_m^1 is a coupled gateway of p_i, and for each $i \in [m+1, a_m]$, if p_i is a via gateway, then q_m^2 is a coupled gateway of p_i. Thus we can set candidate coupled gateways accordingly.

If $a_m = b_k - 1$ and $b_m = a_1 + 1$, then the candidate coupled gateways of all gateways in $[1, k]$ have been computed and we can stop the algorithm. If $a_m = b_k - 1$ but $b_m > a_1 + 1$, we work recursively on the subproblem $([a_1 + 1, b_m - 1], [1, k], V_t(1, k))$ (note that the size of the first interval is reduced by at least half). Similarly, if $b_m = a_1 + 1$ but $a_m < b_k - 1$, then we work recursively on the subproblem $([a_m + 1, b_k - 1], [1, k], V_t(1, k))$. Otherwise, both $b_m > a_1 + 1$ and $a_m < b_k - 1$ hold, and we proceed as follows. We have the following two lemmas that are similar to Lemmas 4 and 5. By definition, either $a_m > b_m$ or $a_m = b_m = m$.

▶ **Lemma 8.** *(1) The path $\pi_{q_m^2}(p_{a_m})$ contains a point in the interior of $R(s)$ only if the last edge of the path intersects the bottom boundary of $R(s)$, in which the intersection at the bottom boundary of $R(s)$ has x-coordinate in $[x(s), x(p_{a_m+1})]$. (2) The path $\pi_{q_m^1}(p_{b_m})$ contains a point in the interior of $R(s)$ only if the last edge of the path intersects the left boundary of $R(s)$, in which case the intersection at the left boundary of $R(s)$ has y-coordinate in $[y(s), y(p_{b_m-1})]$.*

▶ **Lemma 9.** *(1) If the last edge of $\pi_{q_m^2}(p_{a_m})$ intersects the bottom boundary of $R(s)$, then p_i cannot be a via gateway for any $i \in [a_m+1, b_k-1]$. (2) If the last edge of $\pi_{q_m^1}(p_{b_m})$ intersects the left boundary of $R(s)$, then p_i cannot be a via gateway for any $i \in [a_1 + 1, b_m - 1]$.*

In constant time we can check whether the two cases in Lemma 9 happen. If both cases happen, then we can stop the algorithm. If the second case happens and the first one does not, then we recursively work on the subproblem $([a_m + 1, b_k - 1], [1, k], V_t(1, k))$. If the first case happens and the second one does not, then we recursively work on the subproblem $([a_1 + 1, b_m - 1], [1, k], V_t(1, k))$. In the following, we assume that neither case happens. By Lemma 8, neither $\pi_{q_m^2}(p_{a_m})$ nor $\pi_{q_m^1}(p_{b_m})$ contains a point in the interior of $R(s)$. Consequently, we have the following lemma.

▶ **Lemma 10.** *(1) For each $i \in [a_m + 1, b_k - 1]$, if p_i is a via gateway, then p_i has a coupled gateway in $V_t(m, k)$. If $q_m^2 \neq q_k$, then $\pi_{q_m^2}(p_{a_m})$ does not intersect $\pi_{q_k}(p_{b_k})$. (2) For each $i \in [a_1 + 1, b_m - 1]$, if p_i is a via gateway, then p_i has a coupled gateway in $V_t(1, m)$. If $q_m^1 \neq q_1$, then $\pi_{q_m^1}(p_{b_m})$ does not intersect $\pi_{q_1}(p_{a_1})$.*

Based on Lemma 10, our algorithm proceeds as follows. If $q_m^2 = q_k$, then we set q_k as the candidate coupled gateway for each p_i with $i \in [a_m+1, b_k-1]$. Otherwise, we call the algorithm recursively on the subproblem $([a_m + 1, b_k - 1], [m, k], V_t(m, k))$. Similarly, if $q_m^1 = q_1$, then we set q_1 as the candidate coupled gateway for each p_i with $i \in [a_1 + 1, b_m - 1]$. Otherwise, we call the algorithm recursively on the subproblem $([a_1 + 1, b_m - 1], [1, m], V_t(1, m))$.

For the running time, notice that the stair-walking procedure spends $O(1)$ time on finding a coupled gateway for a gateway of $V(s)$. Hence, the overall time of the procedure in the entire algorithm is $O(n_s)$. Consider a subproblem $([i', j'], [i, j], V_t(i, j))$. To solve it, after spending $O(|V_t(i, j)|)$ time, we either reduce the problem to another subproblem in which the first interval is at most half the size of $[i', j']$ and the third gateway set is still $V_t(i, j)$, or reduce it to two sub-problems such that each of them has the first interval at most half the size of $[i', j']$ and the third gateway sets of the two sub-problems are two disjoint subsets of $V_t(i, j)$. Hence, if we consider the algorithm as a tree structure, the height of the tree is $O(\log n_s)$ and the total time we spend on each level of the tree is $O(n_t)$. Therefore, the overall time of the algorithm is $O(n_s + n_t \log n_s)$.

The above describes our algorithm on the gateways of s in the first quadrant of s. We run the same algorithm for all quadrants of s, and for each quadrant, we will find an s-t path. Finally, we return the path with the smallest length as our solution.

▶ **Lemma 11.** *The running time of the query algorithm is $O(\log n + n_s + n_t \log n_s)$.*

4 Reducing the Query Time to $O(\log n)$

Since both n_s and n_t are $O(\log n)$, the query time of Lemma 11 is $O(\log n \log\log n)$. To further reduce it to $O(\log n)$, we need to change our graph G to a slightly larger graph G_1 such that t only needs $O(\log n/\log\log n)$ gateways while s still has $O(\log n)$ gateways, i.e., $n_s = O(\log n)$ and $n_t = O(\log n/\log\log n)$. To this end, we introduce more Steiner points on the cut-lines. A similar idea was also used in [6] to reduce the number of gateways to $O(\sqrt{\log n})$. However, since we are allowed to have more gateways than $O(\sqrt{\log n})$, we do not need as many Steiner points as those in [6], which is the reason why we use less preprocessing.

Specifically, comparing with G, the new graph G_1 has the following changes. As in [6], we first define "super-levels". Recall that the cut-line tree $\mathcal{T}$ has $O(\log n)$ levels (with the root at the first level). We further partition all levels of the tree into $O(\log n/\log\log n)$ super-levels: For any i, the i-th super-level contains the levels from $(i-1)\cdot\log\log n + 1$ to $i\cdot\log\log n$. Hence, each super-level has at most $\log\log n$ levels.

Let u be a node at the highest level of the i-th super level of $\mathcal{T}$. Let $\mathcal{T}_u$ be the sub-tree of $\mathcal{T}$ rooted at u excluding the nodes outside the i-th level (thus $\mathcal{T}_u$ has at most $\log n - 1$ nodes). Recall that u is associated with a subset $\mathcal{V}(u)$ of polygon vertices and each vertex $v \in \mathcal{T}_u$ is associated with a cut-line $l(v)$. For each point $p \in \mathcal{V}(u)$ and each vertex $v \in \mathcal{T}_u$, if p is horizontally visible to $l(v)$, then p defines a *type-3* Steiner point on $l(v)$. In this way, p defines $O(\log n)$ type-3 Steiner points on the cut-lines of $\mathcal{T}_u$ (in contrast, p defines only $O(\log\log n)$ type-2 Steiner points on the cut-lines of $\mathcal{T}_u$ in our original graph G). Hence, each polygon vertex p defines a total of $O(\log^2 n/\log\log n)$ type-3 Steiner points since $\mathcal{T}$ has $O(\log n/\log\log n)$ super-levels. The total number of type-3 Steiner points on all cut-lines is $O(n\log^2 n/\log\log n)$. Note that each type-2 Steiner point in our original graph G becomes a type-3 Steiner point. For convenience of discussion, those type-3 Steiner points of G_1 that are originally type-2 Steiner points of G are also called type-2 Steiner points of G_1.

Type-1 Steiner points are defined in the same way as before, so their number is $O(n)$. We still use $\mathcal{V}_1$ to denote the set of all type-1 Steiner points and all polygon vertices. We use $\mathcal{V}_2$ to denote the set of all type-2 Steiner points of G_1.

The edges of G_1 are defined with respect to all Steiner points in the same way as G. We omit the details. In summary, G_1 has $O(n\log^2 n/\log\log n)$ vertices and edges. Note that the original graph G is a sub-graph of G_1 in that every vertex of G is also a vertex of G_1 and each path of G corresponds to a path in G_1 with the same length.

Consider a query point t. The gateway set $V_g^1(t, G_1)$ is defined the same as before, and thus its size is $O(1)$. Thanks to more Steiner points, the size of $V_g^2(t, G_1)$ can now be reduced to $O(\log n/\log\log n)$. Specifically, $V_g^2(t, G_1)$ is defined as follows (similar to that in [6]).

As in [6], we first define the *relevant projection cut-lines* of t. We only discuss the right side of t, and the left side is symmetric. Recall that t has at most one projection cut-line in each level of $\mathcal{T}$. Among all projection cut-lines that are in the same super-level, the one closest to t is called a *relevant projection cut-line* of t. Since there are $O(\log n/\log\log n)$ super-levels and each super-level has at most one relevant projection cut-line to the right of t, t has $O(\log n/\log\log n)$ relevant projection cut-lines. For each such cut-line l, the type-3 Steiner point (if any) immediately above (resp., below) the horizontal projection t' of t on l is included in $V_g^2(t, G_1)$ if it is visible to t'. Thus, $|V_g^2(t, G_1)| = O(\log n/\log\log n)$.

By Lemma 11, if $|V_g^2(t, G_1)| = O(\log n/\log\log n)$, the query time becomes $O(\log n)$ as long as $|V_g^2(s, G_1)| = O(\log n)$. This implies that for s, we can simply use its original gateway set on type-2 Steiner points, i.e., $V_g^2(s, G_1) = V_g^2(s, G)$. As will be clear later, this will help save time and space in the preprocessing. We also define $V_g^1(s, G_1)$ in the same way as before.

▶ **Lemma 12.** *For any two query points s and t, if there does not exist a trivial shortest s-t path, then there is a shortest s-t path containing a gateway of s and a gateway of t.*

▶ **Lemma 13.** *With $O(n\log^3 n/\log\log n)$ time and $O(n\log^2 n/\log\log n)$ space preprocessing, we can compute $V_g(s,G_1)$ and $V_g(t,G_1)$ in $O(\log n)$ time for any two query points s and t.*

In the preprocessing, for each vertex q of G_1 (which is also considered as a point in $\mathcal{P}$), we compute a shortest path tree $T(q)$ but only for the points in $\mathcal{V}_1 \cup \mathcal{V}_2$ using the algorithm [24, 25]. Since $|\mathcal{V}_1 \cup \mathcal{V}_2| = O(n\log n)$, $T(p)$ has $O(n\log n)$ vertices and can be computed in $O(n\log^2 n)$ time [24, 25]. Since G_1 has $O(n\log^2 n/\log\log n)$ vertices, the preprocessing takes $O(n^2\log^4 n/\log\log n)$ time and $O(n^2\log^3 n/\log\log n)$ space in total.

With Lemma 11 and the new gateway sets, we can reduce the query time to $O(\log n)$.

Using the techniques in [6], we can further reduce the complexities of the preprocessing so that they are functions of h, in addition to $O(n)$, as shown in the following theorem.

▶ **Theorem 14.** *With $O(n + h^2\log^4 h/\log\log h)$ time and $O(n + h^2\log^3 h/\log\log h)$ space preprocessing, given s and t, we can compute their shortest path length in $O(\log n)$ time and an actual shortest s-t path can be output in time linear in the number of edges of the path.*

References

1 A. Aggarwal, M. Klawe, S. Moran, P. Shor, and R. Wilbur. Geometric Applications of a Matrix-Searching Algorithm. *Algorithmica*, 2:195–208, 1987.

2 S.W. Bae and H. Wang. L_1 shortest path queries in simple polygons, 2018. `arXiv:1809.07481`.

3 R. Bar-Yehuda and B. Chazelle. Triangulating disjoint Jordan chains. *International Journal of Computational Geometry and Applications*, 4(4):475–481, 1994.

4 B. Chazelle. Triangulating a Simple Polygon in Linear Time. *Discrete and Computational Geometry*, 6:485–524, 1991.

5 D.Z. Chen, O. Daescu, and K.S. Klenk. On geometric path query problems. *International Journal of Computational Geometry and Applications*, 11(6):617–645, 2001.

6 D.Z. Chen, R. Inkulu, and H. Wang. Two-Point L_1 Shortest Path Queries in the Plane. *Journal of Computational Geometry*, 1:473–519, 2016.

7 D.Z. Chen, K.S. Klenk, and H.-Y.T. Tu. Shortest path queries among weighted obstacles in the rectilinear plane. *SIAM Journal on Computing*, 29(4):1223–1246, 2000.

8 D.Z. Chen and H. Wang. Computing L_1 Shortest Paths Among Polygonal Obstacles in the Plane. *Algorithmica*, 2019. https://doi.org/10.1007/s00453-018-00540-x.

9 D.Z. Chen and J. Xu. Shortest path queries in planar graphs. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 469–478, 2000.

10 Y.-J. Chiang and J.S.B. Mitchell. Two-point Euclidean shortest path queries in the plane. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 215–224, 1999.

11 K. Clarkson, S. Kapoor, and P. Vaidya. Rectilinear shortest paths through polygonal obstacles in $O(n\log^2 n)$ time. In *Proceedings of the 3rd Annual Symposium on Computational Geometry (SoCG)*, pages 251–257, 1987.

12 K. Clarkson, S. Kapoor, and P. Vaidya. Rectilinear shortest paths through polygonal obstacles in $O(n\log^{2/3} n)$ time. Manuscript, 1988.

13 H.N. Djidjev. Efficient algorithms for shortest path queries in planar digraphs. In *Proceedings of the 22nd International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 151–165, 1996.

14 H.A. ElGindy and P. Mitra. Orthogonal shortest route queries among axis parallel rectangular obstacles. *International Journal of Computational Geometry and Application*, 4:3–24, 1994.

15 J. Fakcharoenphol and S. Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *Journal of Computer and System Sciences*, 72:868–889, 2006.

16 P. Gawrychowski, S. Mozes, O. Weimann, and C. Wulff-Nilsen. Better tradeoffs for exact distance oracles in planar graphs. In *Proceedings of the 29rd Annual Symposium on Discrete Algorithms (SODA)*, pages 515–529, 2018.

17 L.J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. *Journal of Computer and System Sciences*, 39(2):126–152, 1989.

18 H. Guo, A. Maheshwari, and J.-R. Sack. Shortest Path Queries in Polygonal Domains. In *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management (AAIM)*, pages 200–211, 2008.

19 J. Hershberger. A new data structure for shortest path queries in a simple polygon. *Information Processing Letters*, 38(5):231–235, 1991.

20 J. Hershberger and J. Snoeyink. Computing minimum length paths of a given homotopy class. *Computational Geometry: Theory and Applications*, 4(2):63–97, 1994.

21 J. Hershberger and S. Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM Journal on Computing*, 28(6):2215–2256, 1999.

22 R. Inkulu and S. Kapoor. Planar rectilinear shortest path computation using corridors. *Computational Geometry: Theory and Applications*, 42(9):873–884, 2009.

23 M.M. Klawe and D.J. Kleitman. An Almost Linear Time Algorithm for Generalized Matrix Searching. *SIAM Journal on Discrete Mathematics*, 3:81–97, 1990.

24 J.S.B. Mitchell. An optimal algorithm for shortest rectilinear paths among obstacles. Abstracts of the *1st Canadian Conference on Computational Geometry*, 1989.

25 J.S.B. Mitchell. L_1 shortest paths among polygonal obstacles in the plane. *Algorithmica*, 8(1):55–88, 1992.

26 S. Mozes and C. Sommer. Exact distance oracles for planar graphs. In *Proceedings of the 23rd Annual Symposium on Discrete Algorithms (SODA)*, pages 209–222, 2012.