A Linear-Time Algorithm for the Geodesic Center of a Simple Polygon

Hee Kap Ahn*[3], Luis Barba[1,2], Prosenjit Bose[1], Jean-Lou De Carufel[1], Matias Korman[4,5], and Eunjin Oh[3]

1 **School of Computer Science, Carleton University, Ottawa, Canada**
`jit@scs.carleton.ca, jdecaruf@cg.scs.carleton.ca`
2 **Département d'Informatique, Université Libre de Bruxelles, Brussels, Belgium**
`lbarbafl@ulb.ac.be`
3 **Department of Computer Science and Engineering, POSTECH, 77 Cheongam-Ro, Nam-Gu, Pohang, Gyeongbuk, Korea**
`{heekap, jin9082}@postech.ac.kr`
4 **National Institute of Informatics (NII), Tokyo, Japan**
`korman@nii.ac.jp`
5 **JST, ERATO, Kawarabayashi Large Graph Project**

Abstract

Let P be a closed simple polygon with n vertices. For any two points in P, the geodesic distance between them is the length of the shortest path that connects them among all paths contained in P. The geodesic center of P is the unique point in P that minimizes the largest geodesic distance to all other points of P. In 1989, Pollack, Sharir and Rote [Disc. & Comput. Geom. 89] showed an $O(n \log n)$-time algorithm that computes the geodesic center of P. Since then, a longstanding question has been whether this running time can be improved (explicitly posed by Mitchell [Handbook of Computational Geometry, 2000]). In this paper we affirmatively answer this question and present a linear time algorithm to solve this problem.

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling

Keywords and phrases Geodesic distance, facility location, 1-center problem, simple polygons

Digital Object Identifier 10.4230/LIPIcs.SOCG.2015.209

1 Introduction

Let P be a simple polygon with n vertices. Given two points x, y in P, the *geodesic path* $\pi(x, y)$ is the shortest path contained in P connecting x with y. If the straight-line segment connecting x with y is contained in P, then $\pi(x, y)$ is a straight-line segment. Otherwise, $\pi(x, y)$ is a polygonal chain whose vertices (other than its endpoints) are reflex vertices of P. We refer the reader to [20] for more information on geodesic paths.

The *geodesic distance* between x and y, denoted by $|\pi(x, y)|$, is the sum of the Euclidean lengths of each segment in $\pi(x, y)$. Throughout this paper, when referring to the distance between two points in P, we mean the geodesic distance between them. To ease the description, we assume that each vertex of P has a unique farthest neighbor. This *general position* condition was also assumed by Aronov et al. [2] and can be obtained by applying a slight perturbation to the positions of the vertices [10].

* The work by H.-K. Ahn and E. Oh was supported by the NRF grant 2011-0030044 (SRC-GAIA) funded by the Korea government (MSIP).

Given a point $x \in P$, a (geodesic) *farthest neighbor* of x, is a point $f_P(x)$ (or simply $f(x)$) of P whose geodesic distance to x is maximized.

Let $F_P(x)$ be the function that maps each $x \in P$ to the distance to a farthest neighbor of x (i.e., $F_P(x) = |\pi(x, f(x))|$). A point $c_P \in P$ that minimizes $F_P(x)$ is called the *geodesic center* of P. Similarly, a point $s \in P$ that maximizes $F_P(x)$ (together with $f(s)$) is called a *geodesic diametral pair* and their distance is known as the *geodesic diameter*. Asano and Toussaint [3] showed that the geodesic center is unique (whereas it is easy to see that several geodesic diametral pairs may exist).

In this paper, we show how to compute the geodesic center of P in $O(n)$ time. Due to lack of space, some proofs are omitted. For a full version of this paper refer to [1].

1.1 Previous Work

Since the early 1980s the problem of computing the geodesic center (and its counterpart, the geodesic diameter) has received a lot of attention from the computational geometry community. Chazelle [7] gave the first algorithm for computing the geodesic diameter (which runs in $O(n^2)$ time using linear space). Afterwards, Suri [25] reduced it to $O(n \log n)$-time without increasing the space constraints. Finally, Hershberger and Suri [14] presented a fast matrix search technique, one application of which is a linear-time algorithm for computing the diameter. The first algorithm for computing the geodesic center was given by Asano and Toussaint [3], and runs in $O(n^4 \log n)$-time. In 1989, Pollack, Sharir, and Rote [23] improved it to $O(n \log n)$ time. Since then, it has been an open problem whether the geodesic center can be computed in linear time (indeed, this problem was explicitly posed by Pollack et al. [23] and later by Mitchell [20, Chapter 27]).

Several variations of these two problems have been considered. Indeed, the same problem has been studied under different metrics. For example, the L_1 geodesic distance [6], the link distance [9, 15, 24] (where we look for the path with the minimum possible number of bends or *links*), or even rectilinear link distance [21, 22] (a variation of the link distance in which only isothetic segments are allowed). The diameter and center of a simple polygon for both the L_1 and rectilinear link metrics can be computed in linear time (whereas $O(n \log n)$ time is needed for the link distance). Another natural extension is the computation of the diameter and center in polygonal domains (i.e., polygons with one or more holes). Polynomial time algorithms are known for both the diameter [4] and center [5], although the running times are significantly larger (i.e., $O(n^{7.73})$ and $O(n^{12+\varepsilon})$, respectively).

1.2 Outline

In order to compute the geodesic center, c_P, Pollack et al. [23] introduce a linear time *chord-oracle*. Given a chord C that splits P into two sub-polygons, this oracle determines which sub-polygon contains c_P. Combining this operation with an efficient search on a triangulation of P, Pollack et al. narrow the search of c_P within a triangle (and find the center using optimization techniques). Their approach however, does not allow them to reduce the complexity of the problem in each iteration, and hence it runs in $\Theta(n \log n)$ time.

The general approach of our algorithm described in Section 6 is similar: partition P into $O(1)$ cells, use an oracle to determine which cell contains c_P, and recurse within the cell. Our approach differs however in two important aspects that allows us to speed-up the algorithm. First, we do not use the chords of a triangulation of P to partition the problem into cells. We use instead a cutting of a suitable set of chords. Secondly, we compute a set Σ of $O(n)$ functions, each defined in a triangular domain contained in P, such that their

upper envelope, ϕ, coincides with F_P. Thus, we can "ignore" the polygon P and focus only on finding the minimum of the function ϕ.

The search itself uses ε-nets and cutting techniques, which guarantee that both the size of the cell containing c_P and the number of functions of Σ defined in it decrease by a constant fraction (and thus leads to an overall linear time algorithm). This search has however two stopping conditions, (1) reach a subproblem of constant size, or (2) find a triangle containing c_P. In the latter case, we show that ϕ is a convex function when restricted to this triangle. Thus, finding its minimum becomes an optimization problem that we solve in Section 7 using cuttings in $\mathbb{R}^3$. The key of this approach lies in the computation of the functions in Σ and their triangular domains. Each function $g \in \Sigma$ is defined in a triangular domain $\triangle$ contained in P and is associated to a particular vertex w of P. Intuitively speaking, g maps points in $\triangle$ to their (geodesic) distance to w. We guarantee that, for each point $x \in P$, there is one function $g \in \Sigma$ defined in a triangle containing x, such that $g(x) = F_P(x)$. To compute these triangles and their corresponding functions, we proceed as follows.

In Section 2, we use the matrix search technique introduced by Hershberger and Suri [14] to decompose the boundary of P, denoted by ∂P, into connected edge-disjoint chains. Each chain is defined by either (1) a consecutive list of vertices that have the same farthest neighbor v (we say that v is *marked* if it has such a chain associated to it), or (2) an edge whose endpoints have different farthest neighbors (such edge is called a *transition edge*).

In Section 3, we consider each transition edge ab of ∂P independently and compute its *hourglass*. Intuitively, the hourglass of ab, H_{ab}, is the region of P between two chains, the edge ab and the chain of ∂P that contains the farthest neighbors of all points in ab. Inspired by a result of Suri [25], we show that the sum of the combinatorial complexities of all hourglasses defined on a transition edge is $O(n)$. (The *combinatorial complexity*—or simply complexity—of a geometric object is the total number of vertices and edges that define it.) In addition, we provide a new technique to compute all these hourglasses in linear time.

In Section 5 we show how to compute the functions in Σ and their respective triangles. We distinguish two cases: (1) Inside each hourglass H_{ab} of a transition edge, we use a technique introduced by Aronov et al. [2] that uses the shortest-path trees of a and b in H_{ab} to construct $O(|H_{ab}|)$ triangles with their respective functions (for more information on shortest-path trees refer to [11]). (2) For each marked vertex v we compute triangles that encode the distance from v. Moreover, we guarantee that these triangles cover every point of P whose farthest neighbor is v. Overall, we compute the $O(n)$ functions of Σ in linear time.

2 Decomposing the boundary

In this section, we decompose ∂P into chains of consecutive vertices that share the same farthest neighbor and edges of P whose endpoints have distinct farthest neighbors.

Using a result from Hershberger and Suri [14], in $O(n)$ time we can compute the farthest neighbor of each vertex of P. Recall that the farthest neighbor of each vertex of P is always a convex vertex of P [3] and is unique by our general position assumption. The (farthest) *Voronoi region* of a vertex v of P is the set of points $R(v) = \{x \in P : F_P(x) = |\pi(x, v)|\}$ (including boundary points).

We mark the vertices of P that are farthest neighbors of at least one vertex of P. Let M denote the set of marked vertices of P (clearly this set can be computed in $O(n)$ time after applying the result of Hershberger and Suri). In other words, M contains all vertices of P whose Voronoi region contains at least one vertex of P. Given a vertex v of P, the vertices of P whose farthest neighbor is v appear contiguously along ∂P [2].

Therefore, after computing all these farthest neighbors, we effectively split the boundary into subchains, each associated with a different vertex of M; see Figure 1.

Given two points x and y on ∂P, let $\partial P(x, y)$ be the polygonal chain that starts at x and follows the boundary of P clockwise until reaching y. We say that three (non-empty) disjoint sets A, B and C contained in ∂P are in *clockwise order* if $B \subset \partial P(a, c)$ for any $a \in A$ and any $c \in C$. (To ease notation, we say that three points $x, y, z \in \partial P$ are in clockwise order if $\{x\}, \{y\}$ and $\{z\}$ are in clockwise order).

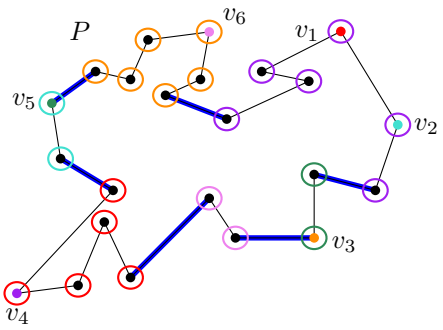

Figure 1 Each vertex of the boundary of P is assigned with a farthest neighbor which is then marked. The boundary is then decomposed into vertex-disjoint chains, each associated with a marked vertex, joined by transition edges (blue) whose endpoints have different farthest neighbors.

Let a and b be the endpoints of a transition edge of ∂P such that b is the clockwise neighbor of a along ∂P. Because ab is a transition edge, we know that $f(a) \neq f(b)$. Recall that we have computed $f(a)$ and $f(b)$ in the previous step and note that $a, b, f(a), f(b)$ are in clockwise order.

For any vertex $v \in \partial P$ such that $f(a) \neq v \neq f(b)$ and $f(a), v, f(b)$ are in clockwise order, we know that there cannot be a vertex u of P such that $f(u) = v$. As proved by Aronov et al. [2, Corollary 2.7.4], if there is a point x on ∂P whose farthest neighbor is v, then x must lie on the open segment (a, b). In other words, the Voronoi region $R(v)$ restricted to ∂P is contained in (a, b).

3 Hourglasses

For any polygonal chain $C = \partial P(p_0, p_k)$, the *hourglass* of C, denoted by H_C, is the simple polygon contained in P bounded by C, $\pi(p_k, f(p_0))$, $\partial P(f(p_0), f(p_k))$ and $\pi(f(p_k), p_0)$; see Figure 2. We call C and $\partial P(f(p_0), f(p_k))$ the *top* and *bottom* chains of H_C, respectively, while $\pi(p_k, f(p_0))$ and $\pi(f(p_k), p_0)$ are referred to as the *walls* of H_C. We say that the hourglass H_C is *open* if its walls are vertex-disjoint. We say C is a *transition chain* if $f(p_0) \neq f(p_k)$ and neither $f(p_0)$ nor $f(p_k)$ are interior vertices of C. In particular, if an edge ab of ∂P is a transition chain, we say that it is a *transition edge* (see Figure 2).

▶ **Lemma 1** (Restatement of Lemma 3.1.3 of [2]). *If C is a transition chain of ∂P, then the hourglass H_C is an open hourglass.*

In the remainder of the paper, all the hourglasses considered are defined by a transition chain. That is, they are open and their top and bottom chains are edge-disjoint.

The following lemma is depicted in Figure 2 and is a direct consequence of the Ordering Lemma proved by Aronov et al. [2, Corollary 2.7.4].

▶ **Lemma 2.** *Let C_1, C_2, C_3 be three edge-disjoint transition chains of ∂P in clockwise order. Then, the bottom chains of H_{C_1}, H_{C_2} and H_{C_3} are also edge-disjoint and are in clockwise order.*

Let γ be a geodesic path joining two points on the boundary of P. We say that γ *separates* two points x_1 and x_2 of ∂P if the points of $X = \{x_1, x_2\}$ and the endpoints of γ alternate along the boundary of P (x_1 and x_2 could coincide with the endpoints of γ in degenerate

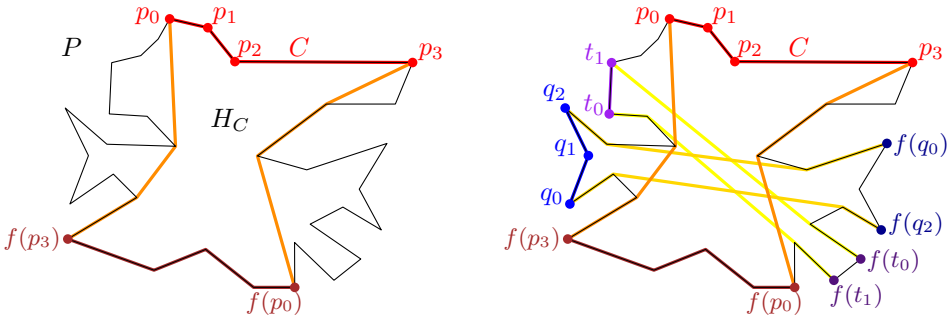

Figure 2 Given two edge-disjoint transition chains, their hourglasses are open and the bottom chains of their hourglasses are also edge-disjoint. Moreover, these bottom chains appear in the same cyclic order as the top chains along ∂P.

cases). We say that a geodesic path γ *separates* an hourglass H if it separates the points of its top chain from those of its bottom chain.

▶ **Lemma 3.** *Let $C_1, \ldots, C_r$ be edge-disjoint transition chains of ∂P. Then, there is a set of $t \leq 10$ geodesic paths $\gamma_1, \ldots, \gamma_t$ with endpoints on ∂P such that for each $1 \leq i \leq r$ there exists $1 \leq j \leq t$ such that γ_j separates H_{C_i}. Moreover, this set can be computed in $O(n)$ time.*

A *chord* of P is an edge joining two non-adjacent vertices a and b of P such that $ab \subseteq P$. Therefore, a chord splits P into two sub-polygons.

▶ **Lemma 4** (Restatement of Lemma 3.4.3 of [2])**.** *Let $C_1, \ldots, C_r$ be a set of edge-disjoint transition chains of ∂P in clockwise order. Then each chord of P appears in $O(1)$ hourglasses among $H_{C_1}, \ldots, H_{C_r}$.*

▶ **Lemma 5.** *Let x, u, y, v be four vertices of P in clockwise order. Given the shortest-path trees T_x and T_y of x and y in P, respectively, such that T_x and T_y can answer lowest common ancestor (LCA) queries in $O(1)$ time, we can compute the path $\pi(u, v)$ in $O(|\pi(u, v)|)$ time. Moreover, all edges of $\pi(u, v)$, except perhaps one, belong to $T_x \cup T_y$.*

▶ **Lemma 6.** *Let P be a simple polygon with n vertices. Given k disjoint transition chains $C_1, \ldots, C_k$ of ∂P, it holds that*

$$\sum_{i=1}^{k} |H_{C_i}| = O(n).$$

Proof. Because the given transition chains are edge-disjoint, Lemma 2 implies that the bottom chains of their respective hourglasses are also edge-disjoint. Therefore, the sum of the complexities of all the top and bottom chains of these hourglasses is $O(n)$. To bound the complexity of their walls we use Lemma 4. Since no chord is used more than a constant number of times, it suffices to show that the total number of chords used by all these hourglasses is $O(n)$.

To prove this, we use Lemma 3 to construct $O(1)$ *splitting chains* $\gamma_1, \ldots, \gamma_t$ such that for each $1 \leq i \leq k$, there is a splitting chain γ_j that separates the top and bottom chains of H_{C_i}. For each $1 \leq j \leq t$, let $\mathcal{H}^j = \{H_{C_i} : \text{the top and bottom chain of } H_{C_i} \text{ are separated by } \gamma_j\}$. Since the complexity of the shortest-path trees of the endpoints of γ_j is $O(n)$ [11], and from the fact that the chains $C_1, \ldots, C_k$ are edge-disjoint, Lemma 5 implies that the total number

of edges in all the hourglasses of $\mathcal{H}^j$ is $O(n)$. Moreover, because each of these edges appears in $O(1)$ hourglasses among $C_1, \ldots, C_k$, we conclude that

$$\sum_{H \in \mathcal{H}^j} |H| = O(n).$$

Since we have only $O(1)$ splitting chains, our result follows. ◀

3.1 Building hourglasses

Let E be the set of transition edges of ∂P. Given a transition edge $ab \in E$, we say that H_{ab} is a *transition hourglass*. In this section, we present an algorithm that computes each transition hourglass of P in $O(n)$ time.

By Lemma 3 we can compute a set of $O(1)$ separating paths such that for each transition edge ab, the transition hourglass H_{ab} is separated by one (or more) paths in this set. For each endpoint of the $O(1)$ separating paths we compute its shortest-path tree in linear time [8, 11]. In addition, we preprocess these trees in linear time to support LCA queries [13]. Both computations need linear time per endpoint and use $O(n)$ space. Since we do this process for a constant number of endpoints, overall this preprocessing takes $O(n)$ time.

Let γ be a separating path. Note that γ separates the boundary of P into two chains S and S' such that $S \cup S' = \partial P$. Let $\mathcal{H}(\gamma)$ be the set of transition hourglasses separated by γ whose transition edge is contained in S (whenever an hourglass is separated by more than one path, we pick one arbitrarily). Note that we can classify all transition hourglasses into the sets $\mathcal{H}(\gamma)$ in $O(n)$ time (since $O(1)$ separating paths are considered).

We claim that we can compute all transition hourglasses of $\mathcal{H}(\gamma)$ in $O(n)$ time. By construction, the wall of each of these hourglasses consists of a (geodesic) path that connects a point in S with a point in S'. Let $u \in S$ and $v \in S'$ be two vertices such that $\pi(u, v)$ is the wall of a hourglass in $\mathcal{H}(\gamma)$. Because LCA queries can be answered in $O(1)$ time [13], Lemma 5 allows us to compute this path in $O(|\pi(u, v)|)$ time. Therefore, we can compute all hourglasses of $\mathcal{H}(\gamma)$ in $O(\sum_{H \in \mathcal{H}(\gamma)} |H| + n) = O(n)$ time by Lemma 6. Because only $O(1)$ separating paths are considered, we obtain the following result.

▶ **Lemma 7.** *The total complexity of the transition hourglasses of all transition edges of P is $O(n)$. Moreover, all these hourglasses can be constructed in $O(n)$ time.*

4 Funnels

Let $C = (p_0, \ldots, p_k)$ be a chain of ∂P and let v be a vertex of P not in C. The *funnel* of v to C, denoted by $S_v(C)$, is the simple polygon bounded by C, $\pi(p_k, v)$ and $\pi(v, p_0)$; see Figure 3 (*a*). Note that the paths $\pi(v, p_k)$ and $\pi(v, p_0)$ may coincide for a while before splitting into edge-disjoint chains. A subset $R \subset P$ is *geodesically convex* if for every $x, y \in R$, the path $\pi(x, y)$ is contained in R. This funnel $S_v(C)$ is then the minimum geodesically convex set that contains v and C. See Lee and Preparata [16] or Guibas et al. [11] for more details on funnels.

▶ **Lemma 8.** *Let v be a vertex of P and let C be a transition chain such that $R(v) \cap \partial P \subseteq C$ and $v \notin C$. Then, $R(v)$ is contained in the funnel $S_v(C)$*

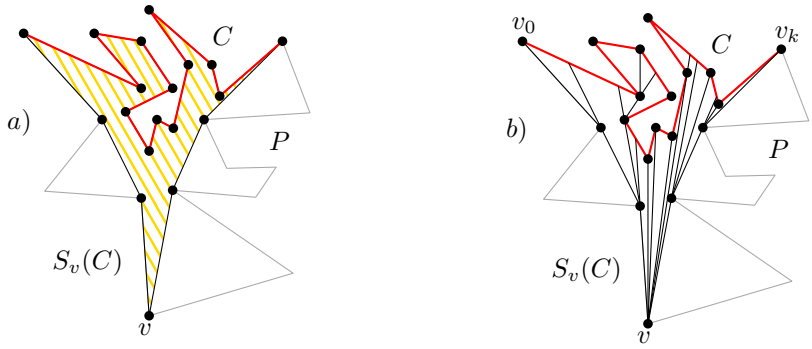

Figure 3 *a*) The funnel $S_v(C)$ of a vertex v and a chain C contained in ∂P are depicted. *b*) The decomposition of $S_v(C)$ into apexed triangles produced by the shortest-path map of v.

4.1 Funnels of marked vertices

Recall that for each marked vertex $v \in M$, we know at least of one vertex on ∂P such that v is its farthest neighbor.

▶ **Lemma 9.** *Let x be a point in P. If $f(x) = v$ for some marked vertex $v \in M$, then $x \in S_v(C_v)$.*

For any marked vertex v, let $u_1, \ldots, u_{k-1}$ be the vertices of P such that $v = f(u_i)$ and assume that $u_1, \ldots, u_{k-1}$ are in clockwise order. Let u_0 and u_k be the neighbors of u_1 and u_{k-1} other than u_2 and u_{k-2}, respectively. Note that both u_0u_1 and $u_{k-1}u_k$ are transition edges of P. Thus, we can assume that their transition hourglasses have been computed.

Let $C_v = (u_0, \ldots, u_k)$ and consider the funnel $S_v(C_v)$. We call C_v the *main chain* of $S_v(C_v)$ while $\pi(u_k, v)$ and $\pi(v, u_0)$ are referred to as the *walls* of the funnel. Because $v = f(u_1) = f(u_{k-1})$, we know that v is a vertex of both $H_{u_0u_1}$ and $H_{u_{k-1}u_k}$. By definition, we have $\pi(v, u_0) \subset H_{u_0u_1}$ and $\pi(v, u_k) \subset H_{u_{k-1}u_k}$. Thus, we can explicitly compute both paths $\pi(v, u_0)$ and $\pi(v, u_k)$ in $O(|H_{u_0u_1}| + |H_{u_{k-1}u_k}|)$ time. So, overall, the funnel $S_v(C_v)$ can be constructed in $O(k + |H_{u_0u_1}| + |H_{u_{k-1}u_k}|)$ time. Recall that, by Lemma 6, the total sum of the complexities of the transition hourglasses is $O(n)$. In particular, we can bound the total time needed to construct the funnels of all marked vertices by $O(n)$.

Since the complexity of the walls of these funnels is bounded by the complexity of the transition hourglasses used to compute them, by Lemma 7 we get that

$$\sum_{v \in M} |S_v(C_v)| = O\left(n + \sum_{ab \in E} |H_{ab}|\right) = O(n).$$

▶ **Lemma 10.** *The total complexity of the funnels of all marked vertices of P is $O(n)$. Moreover, all these funnels can be constructed in $O(n)$ time.*

5 Covering the polygon with apexed triangles

An *apexed triangle* $\triangle = (a, b, c)$ with *apex* a is a triangle contained in P with an associated distance function $g_\triangle(x)$, called the *apex function* of $\triangle$, such that (1) a is a vertex of P, (2) $b, c \in \partial P$, and (3) there is a vertex w of P, called the *definer* of $\triangle$, such that

$$g_\triangle(x) = \begin{cases} -\infty & \text{if } x \notin \triangle\ , \\ |xa| + |\pi(a, w)| = |\pi(x, w)| & \text{if } x \in \triangle\ . \end{cases}$$

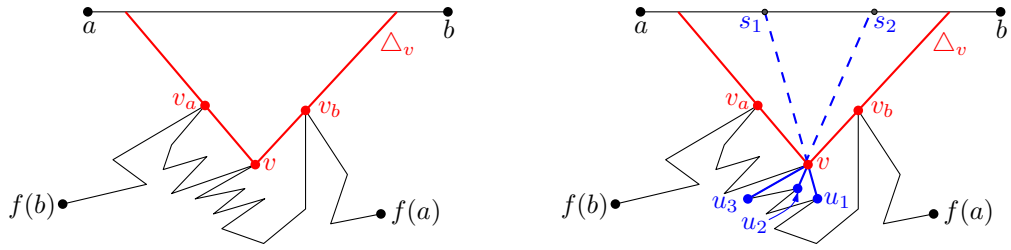

Figure 4 (left) A vertex v visible from the segment ab lying on the bottom chain of H_{ab}, and the triangle $\triangle_v$ which contains the portion of ab visible from v. (right) The children u_1 and u_2 of v are visible from ab while w_3 is not. The triangle $\triangle_v$ is split into apexed triangles by the rays going from u_1 and u_2 to v.

In this section, we show how to find a set of $O(n)$ apexed triangles of P such that the upper envelope of their apex functions coincides with $F_P(x)$. To this end, we first decompose the transition hourglasses into apexed triangles that encode all the geodesic distance information inside them. For each marked vertex $v \in M$, we construct a funnel that contains the Voronoi region of v. We then decompose this funnel into apexed triangles that encode the distance from v.

5.1 Inside the transition hourglass

Let ab be a transition edge of P such that b is the clockwise neighbor of a along ∂P. Let B_{ab} denote the open bottom chain of H_{ab}. As noticed above, a point on ∂P can be farthest from a vertex in B_{ab} only if it lies in the open segment ab. That is, if v is a vertex of B_{ab} such that $R(v) \neq \emptyset$, then $R(v) \cap \partial P \subset ab$. In fact, not only is this Voronoi region inside H_{ab} when restricted to the boundary of P, but we can further bound its location and show that $R(v) \subset H_{ab}$. The next result follows directly from Lemma 8.

▶ **Corollary 11.** *Let v be a vertex of B_{ab}. If $R(v) \neq \emptyset$, then $R(v) \subset H_{ab}$.*

Our objective is to compute $O(|H_{ab}|)$ apexed triangles contained in H_{ab}, each with its distance function, such that the upper envelope of these apex functions coincides with $F_P(x)$ restricted to H_{ab} where it "matters".

The same approach was already used by Pollack et al. in [23, Section 3]. Given a segment contained in the interior of P, they show how to compute a linear number of apexed triangles such that $F_P(x)$ coincides with the upper envelope of the corresponding apex functions in the given segment. While the construction we follow is analogous, we use it in the transition hourglass H_{ab} instead of the full polygon P. Therefore, we have to specify what is the relation between the upper envelope of the computed functions and $F_P(x)$. We will show that the upper envelope of the apex functions computed in H_{ab} coincides with $F_P(x)$ inside the Voronoi region $R(v)$ of every vertex $v \in B_{ab}$.

Let T_a and T_b be the shortest-path trees in H_{ab} from a and b rooted at a and b, respectively. We can compute these trees in $O(|H_{ab}|)$ time [11]. For each vertex v such that $f(a), v$ and $f(b)$ are in clockwise order, let v_a and v_b be the neighbors of v in the paths $\pi(v, a)$ and $\pi(v, b)$, respectively. We say that a vertex v is *visible* from ab if $v_a \neq v_b$. Note that if a vertex is visible, then the extension of these segments must intersect the top segment ab. Therefore, for each visible vertex v, we obtain a triangle $\triangle_v$ as shown in Figure 4.

We further split $\triangle_v$ into a series of triangles with apex at v as follows: Let u be a child of v in either T_a or T_b. As noted by Pollack et al., v can be of three types, either (1) u is not

visible from ab (and is hence a child of v in both T_a and T_b); or (2) u is visible from ab, is a child of v only in T_b, and v_bvu is a left turn; or (3) u is visible from ab, is a child of v only in T_a, and v_avu is a right turn.

Let $u_1, \ldots, u_{k-1}$ be the children of v of type (2) sorted in clockwise order around v. Let $c(v)$ be the maximum distance from v to any invisible vertex in the subtrees of T_a and T_b rooted at v; if no such vertex exists, then $c(v) = 0$. Define a function $d_l(v)$ on each vertex v of H_{ab} in a recursive fashion as follows: If v is invisible from ab, then $d_l(v) = c(v)$. Otherwise, let $d_l(v)$ be the maximum of $c(v)$ and $\max\{d_l(u_i) + |u_iv| : u_i \text{ is a child of } v \text{ of type (2)}\}$. Symmetrically, we define a function $d_r(v)$ using the children of type (3) of v.

For each $1 \leq i \leq k-1$, extend the segment u_iv passed v until it intersects ab at a point s_i. Let s_0 and s_k be the intersections of the extensions of vv_a and vv_b with the segment ab. We define k apexed triangles contained in $\triangle_v$ as follows. For each $0 \leq i \leq k-1$, consider the triangle $\triangle(s_i, v, s_{i+1})$ whose associated apexed (left) function is

$$f_i(x) = \begin{cases} |xv| + \max_{j>i}\{c(v), |vu_j| + d_l(u_j)\} & \text{if } x \in \triangle(s_i, v, s_{i+1}) \ , \\ -\infty & \text{otherwise} \ . \end{cases}$$

In a symmetric manner, we define a set of apexed triangles induced by the type (3) children of v and their respective apexed (right) functions.

Let $g_1, \ldots, g_r$ and $\triangle_1, \ldots, \triangle_r$ respectively be an enumeration of all the generated apex functions and apexed triangles such that g_i is defined in the triangle $\triangle_i$. Because each function is determined uniquely by a pair of adjacent vertices in T_a or in T_b, and since these trees have $O(|H_{ab}|)$ vertices, we conclude that $r = O(|H_{ab}|)$.

Note that for each $1 \leq i \leq r$, the apexed triangle $\triangle_i$ has two vertices on the segment ab and a third vertex, say a_i, being its apex such that for each $x \in \triangle_i$, $g_i(x) = |\pi(x, w_i)|$ for some vertex w_i of H_{ab}. Recall that w_i is called the definer of $\triangle_i$. Intuitively, $\triangle_i$ defines a portion of the geodesic distance function from w_i in a constant complexity region.

▶ **Lemma 12.** *Given a transition edge ab of P, we can compute a set $\mathcal{A}_{ab}$ of $O(|H_{ab}|)$ apexed triangles in $O(|H_{ab}|)$ time with the property that for any point $p \in P$ such that $f(p) \in B_{ab}$, there is an apexed triangle $\triangle \in \mathcal{A}_{ab}$ with apex function g and definer equal to $f(p)$ such that*

1. *$p \in \triangle$ and*
2. *$g(p) = F_P(p)$.*

In other words, Lemma 12 says that no information on farthest neighbors is lost if we only consider the functions of $\mathcal{A}_{ab}$ within H_{ab}. In the next section we construct a set of apexed triangles (and their corresponding apex functions), so as to encode the distance from the vertices of M.

5.2 Inside the funnels of marked vertices

We now proceed to split a given funnel into $O(|S_v(C_v)|)$ apexed triangles that encode the distance function from v. To this end, we use the algorithm described by Guibas et al. [12, Section 2] to compute the shortest-path map of v in $S_v(C_v)$ in $O(|S_v(C_v)|)$ time. This algorithm produces a partition of $S_v(C_v)$ into $O(|S_v(C_v)|)$ interior disjoint triangles with vertices on ∂P, such that each triangle consists of all points in $S_v(C_v)$ whose shortest path to v consists of the same sequence of vertices; see Figure 3 (b). Let $\triangle$ be a triangle in this partition and let a be its apex, i.e., the first vertex found along each path $\pi(x, v)$, where

$x \in \triangle$. We define the apex function $g_\triangle(x)$ of $\triangle$ as follows:

$$g_\triangle(x) = \begin{cases} |xa| + |\pi(a,v)| & \text{if } x \in \triangle\ , \\ -\infty & \text{otherwise}\ . \end{cases}$$

Therefore, for each $x \in \triangle$, $g_\triangle(x) = |\pi(x,v)|$.

▶ **Lemma 13.** *The shortest-path map of v in $S_v(C_v)$ can be computed in $O(|S_v(C_v)|)$ time and produces $O(|S_v(C_v)|)$ interior disjoint apexed triangles such that their union covers $S_v(C_v)$. Moreover, for each point $x \in R(v)$, there is an apexed triangle $\triangle$ with apex function $g(x)$ such that (1) $x \in \triangle$ and (2) $g(x) = F_P(x)$.*

Proof. The above procedure splits $S_v(C_v)$ into $O(|S_v(C_v)|)$ apexed triangles, such that the apex function in each of them is defined as the geodesic distance to v. By Lemma 9, if $x \in R(v)$, then $x \in S_v(C_v)$. Therefore, there is an apexed triangle $\triangle$ with apex function $g(x)$ such that $x \in \triangle$ and $g(x) = |\pi(x,v)| = F_P(x)$. Thus, we obtain properties (1) and (2). ◀

6 Prune and search

With the tools introduced in the previous sections, we can describe a prune and search algorithm to compute the geodesic center. The idea of the algorithm is to partition P into $O(1)$ cells using ε-nets, determine in which cell of P the center lies and recurse on that cell as a new subproblem with smaller complexity.

We can discard all apexed triangles that do not intersect the new cell containing the center. Using cuttings to produce this partition of P, we can show that both the complexity of the cell containing the center, and the number of apexed triangles that intersect it decrease by a constant fraction in each iteration of the algorithm. This process is then repeated until either of the two objects has constant descriptive size.

Let τ be the set of all apexed triangles computed in previous sections. Lemmas 6 and 12 bound the number of apexed triangles constructed inside the transition hourglasses, while Lemmas 10 and 13 do so inside the funnels of the marked vertices. We obtain the following.

▶ **Corollary 14.** *The set τ consists of $O(n)$ apexed triangles.*

Let $\phi(x)$ be the upper envelope of the apex functions of the triangles in τ (i.e., $\phi(x) = \max\{g(x) : \triangle \in \tau$ and $g(x)$ is the apex function of $\triangle\}$). The following result is a direct consequence of Lemmas 12 and 13, and shows that the $O(n)$ apexed triangles of τ not only cover P, but their apex functions suffice to reconstruct the function $F_P(x)$.

▶ **Lemma 15.** *The functions $\phi(x)$ and $F_P(x)$ coincide in the domain of points of P, i.e., for each $p \in P$, $\phi(p) = F_P(p)$.*

Given a chord C of P, a *half-polygon* of P is one of the two simple polygons in which C splits P. A *k-cell* of P is a simple polygon obtained as the intersection of at most k half-polygons. Because a k-cell is the intersection of geodesically convex sets, it is also geodesically convex. The recursive algorithm described in this section takes as input a 4-cell R (initially equal to P) containing the geodesic center of P and the set of apexed triangles of τ that intersect R. In each iteration, it produces a new 4-cell of smaller complexity that intersects just a fraction of the apexed triangles and contains the geodesic center of P. By recursing on this new cell, the complexity of the problem is reduced in each iteration.

Let R be a 4-cell of P containing the geodesic center of P and let τ_R be the set of apexed triangles of τ that intersect R. Let $m_R = \max\{|R|, |\tau_R|\}$, where $|R|$ denotes the

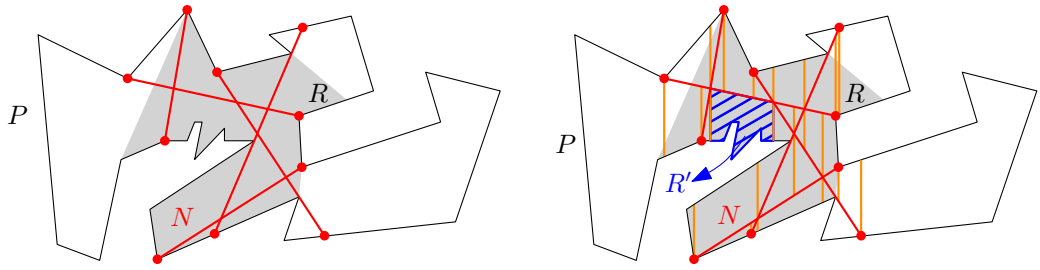

Figure 5 The ϵ-net N splits R into $O(1)$ sub-polygons that are further refined into a 4-cell decomposition using $O(1)$ ray-shooting queries from the vertices of the arrangement defined by N.

combinatorial complexity of R. Recall that, by construction of the apexed triangles, for each triangle of τ_R at least one and at most two of its boundary segments are chords of P. Let $\mathcal{C}$ be the set containing all chords that belong to the boundary of a triangle of τ_R. Therefore, $|\mathcal{C}| \leq 2|\tau_R| \leq 2m_R$.

To construct ε-nets, we need some definitions (for more information on ε-nets refer to [18]). Let φ be the set of all open 4-cells of P. For each $t \in \varphi$, let $\mathcal{C}_t = \{C \in \mathcal{C} : C \cap t \neq \emptyset\}$ be the set of chords of $\mathcal{C}$ induced by t. Finally, let $\varphi_{\mathcal{C}} = \{\mathcal{C}_t : t \in \varphi\}$ be the family of subsets of $\mathcal{C}$ induced by φ. Consider the set system $(\mathcal{C}, \varphi_{\mathcal{C}})$ (denoted by $(\mathcal{C}, \varphi)$ for simplicity).

Let $\varepsilon > 0$ (the exact value of ε will be specified later). Because the VC-dimension of the set system $(\mathcal{C}, \varphi)$ is finite [1], we can compute an ε-net N of $(\mathcal{C}, \varphi)$ in $O(|\mathcal{C}|/\varepsilon) = O(m_R)$ time [18]. The size of N is $O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon}) = O(1)$ and its main property is that any 4-cell that does not intersect a chord of N will intersect at most $\varepsilon|\mathcal{C}|$ chords of $\mathcal{C}$.

Observe that N partitions R into $O(1)$ sub-polygons (not necessarily 4-cells). We further refine this partition to obtain 4-cells. That is, we shoot vertical rays up and down from each endpoint of N, and from the intersection point of any two segments of N, see Figure 5. Overall, this partitions R into $O(1)$ 4-cells such that each either (i) is a convex polygon contained in P of at most four vertices, or otherwise (ii) contains some chain of ∂P. Since $|N| = O(1)$, the whole decomposition can be computed in $O(m_R)$ time (the intersections between segments of N are done in constant time, and for the ray shooting operations we walk along the boundary of R once).

In order to determine which 4-cell contains the geodesic center of P, we extend each edge of a 4-cell to a chord C. This can be done with two ray-shooting queries (each of which takes $O(m_R)$ time). We then use the chord-oracle from Pollack et al. [23, Section 3] to decide which side of C contains c_P. The only requirement of this technique is that the function $F_P(x)$ coincides with the upper envelope of the apex functions when restricted to C, which is true by Lemma 15 and from the fact that τ_R consists of all the apexed triangles of τ that intersect R. Because the chord-oracle described by Pollack et al. [23, Section 3] runs in time linear in the number of functions defined on C, we can decide in total $O(m_R)$ time in which side of C the geodesic center of P lies. Since our decomposition into 4-cells has constant complexity, we need to perform $O(1)$ calls to the oracle before determining the 4-cell R' that contains the geodesic center of P.

The chord-oracle computes the minimum of $F_P(x)$ restricted to the chord before determining the side containing the minimum. In particular, if c_P lies on any chord bounding R', then the chord-oracle will find it. Therefore, we can assume that c_P lies in the interior of R'. Moreover, since N is a ε-net, we know that at most $\varepsilon|\mathcal{C}|$ chords of $\mathcal{C}$ intersect R'.

We can show that the complexity of R' also decreases: since $|\mathcal{C}| \leq 2|\tau_R| \leq 2m_R$, at most $2\varepsilon m_R$ apexed triangles intersect R'. Because $F_P(x)$ is defined in each point of R', Lemma 15

implies that each vertex of R' is covered by at least one apexed triangle of τ_R. Since each apexed triangle can cover at most three vertices, by the pigeonhole principle we conclude that R' can have at most $6\varepsilon m_R$ vertices. Otherwise, an apexed triangle would contain at least four vertices of R'. Thus, if we choose $\varepsilon = 1/12$, we guarantee that both the size of the 4-cell R' and the number of apexed triangles in $\tau_{R'}$ are at most $m_R/2$.

In order to proceed with the algorithm on R' recursively, we need to compute the set $\tau_{R'}$ with the at most $\varepsilon|\mathcal{C}|$ apexed triangles of τ_R that intersect R' (i.e., prune the apexed triangles that do not intersect with R'). For each apexed triangle $\triangle \in \tau_R$, we can determine in constant time if it intersects R' (either one of the endpoints is in $R' \cap \partial P$ or the two boundaries have non-empty intersection in the interior of P). Overall, we need $O(m_R)$ time to compute the at most $\varepsilon|\mathcal{C}|$ triangles of τ_R that intersect R'.

By recursing on R', we guarantee that after $O(\log m_R)$ iterations, we reduce the size of either τ_R or R' to constant. In the former case, the minimum of $F_P(x)$ can be found by explicitly constructing ϕ in $O(1)$ time. In the latter case, we triangulate R' and apply the chord-oracle to determine which triangle will contain c_P. The details needed to find the minimum of $\phi(x)$ inside this triangle are given in the next section.

▶ **Lemma 16.** *In $O(n)$ time we can find either the geodesic center of P or a triangle containing the geodesic center.*

7 Finding the center within a triangle

In order to complete the algorithm it remains to show how to find the geodesic center of P for the case in which R' is a triangle. If this triangle is in the interior of P, it may happen that several apexed triangles of τ fully contain R'. Thus, the pruning technique used in the previous section cannot be further applied. We solve this case with a different approach.

Recall that $\phi(x)$ denotes the upper envelope of the apex functions of the triangles in τ, and the geodesic center is the point that minimizes ϕ. The key observation is that, as it happened with chords, the function $\phi(x)$ restricted to R' is convex.

Let $\triangle_1, \triangle_2, \ldots, \triangle_m$ be the set of $m = O(n)$ apexed triangles of τ that intersect R'. Let a_i and w_i be the apex and the definer of $\triangle_i$, respectively. Let $g_i(x)$ be the apex function of $\triangle_i$ such that

$$g(x) = \begin{cases} |xa_i| + \kappa_i & \text{if } x \in \triangle_i \ , \\ -\infty & \text{otherwise} \ , \end{cases}$$

where $\kappa_i = |\pi(a_i, w_i)|$ is a constant.

By Lemma 15, $\phi(x) = F_P(x)$. Therefore, the problem of finding the center is equivalent to the following optimization problem in $\mathbb{R}^3$:

(P1). Find a point $(x, r) \in \mathbb{R}^3$ minimizing r subject to $x \in R'$ and

$$g_i(x) \leq r, \text{ for } 1 \leq i \leq m.$$

Thus, we need only to find the solution to (P1) to find the geodesic center of P. We use some remarks described by Megiddo in order to simplify the description of (P1) [19].

To simplify the formulas, we square the equation $|xa_i| \leq r - \kappa_i$:

$$\|x\|^2 - 2x \cdot a_i + \|a_i\|^2 = |xa_i|^2 \leq (r - \kappa_i)^2 = r^2 - 2r\kappa_i + \kappa_i^2.$$

And finally for each $1 \leq i \leq m$, we define the function $h_i(x, r)$ as follows:

$$h_i(x, r) = \begin{cases} \|x\|^2 - 2x \cdot a_i + \|a_i\|^2 - r^2 + 2r\kappa_i - \kappa_i^2 & \text{if } x \in \triangle_i \ , \\ -\infty & \text{otherwise} \ . \end{cases}$$

Therefore, our optimization problem can be reformulated as:

(P2). Find a point $(x, r) \in \mathbb{R}^3$ such that r is minimized subject to $x \in R'$ and

$$h_i(x, r) \leq 0 \text{ and } r > \max\{\kappa_i\}, \text{ for } 1 \leq i \leq m.$$

Let $h'_i(x, r) = \|x\|^2 - 2x \cdot a_i + \|a_i\|^2 - r^2 + 2r\kappa_i - \kappa_i^2$ be a function defined in the entire plane and let (P2′) be an optimization problem analogous to (P2) where every instance of $h_i(x, r)$ is replaced by $h'_i(x, r)$. The optimization (P2′) was studied by Megiddo in [19]. We provide some of the intuition used by Megiddo to solve this problem.

Although the functions $h'_i(x, r)$ are not linear, they all have the same non-linear terms. Therefore, for $i \neq j$, we get that $h'_i(x, r) = h'_j(x, r)$ defines a *separating plane*

$$\gamma_{i,j} = \{(x, r) \in \mathbb{R}^3 : 2(\kappa_i - \kappa_j)r - 2(a_i - a_j) \cdot x + \|a_i\|^2 - \|a_j\|^2 - \kappa_i^2 + \kappa_j^2 = 0\} .$$

As noted by Megiddo [19], this separating plane has the following property: If the solution (x, r) to (P2′) is known to lie to one side of $\gamma_{i,j}$, then we know that one of the constraints is redundant. Thus, to solve (P2′) it sufficed to have a *side-decision oracle* to determine in which side of a plane $\gamma_{i,j}$ the solution lies. Megiddo showed how to implement this oracle in a way that the running time is proportional to the number of constraints [19].

Once we have such an oracle, Megiddo's problem can be solved using a prune and search approach: pair the functions arbitrarily, and consider the set of $m/2$ separating planes defined by these pairs. For some constant t, compute a $1/t$-cutting in $\mathbb{R}^3$ of the separating planes. A $1/t$-cutting is a partition of the plane into $O(t^3) = O(1)$ convex regions each of which is of constant complexity and intersects at most $m/2t$ separating planes. A cutting of planes can be computed in linear time in $\mathbb{R}^3$ for any $t = O(1)$ [17]. After computing the cutting, determine in which of the regions the minimum lies by performing $O(1)$ calls to the side-decision oracle. Because at least $(t - 1)m/2t$ separating planes do not intersect this constant complexity region, for each of them we can discard one of the constraints as it becomes redundant. Repeating this algorithm recursively we obtain a linear running time.

To solve (P2) we follow a similar approach, but our set of separating planes needs to be extended in order to handle apex functions as they are only defined in the same way as in (P2′) in a triangular domain. Note that no vertex of an apexed triangle can lie inside R'.

7.1 Optimization problem in a convex domain

In this section we describe our algorithm to solve the optimization problem (P2). To this end, we pair the apexed triangles arbitrarily to obtain $m/2$ pairs. By identifying the plane where P lies with the plane $Z_0 = \{(x, y, z) : z = 0\}$, we can embed each apexed triangle in $\mathbb{R}^3$. A *plane-set* is a set consisting of at most five planes in $\mathbb{R}^3$. For each pair of apexed triangles $(\triangle_i, \triangle_j)$ we define its plane-set as follows: For each chord of P bounding either $\triangle_i$ or $\triangle_j$ (at most two chords on each triangle), consider the line extending this chord and the vertical extrusion of this line in $\mathbb{R}^3$, i.e., the plane containing this chord orthogonal to Z_0. Moreover, consider the separating plane $\gamma_{i,j}$. The set containing these planes is the plane-set of the pair $(\triangle_i, \triangle_j)$.

Let Γ be the union of all the plane-sets defined by the $m/2$ pairs of apexed triangles. Because the plane-set of each pair $(\triangle_i, \triangle_j)$ consists of at most five planes and contains at least one plane unique to this pair, say $\gamma_{i,j}$, we infer that $m/2 \leq |\Gamma| \leq 5m/2$.

Compute a $1/t$-cutting of Γ in $O(m)$ time for some constant t to be specified later. Because t is constant, this $1/t$-cutting splits the space into $O(1)$ convex regions, each bounded by a

constant number of planes [17]. Using a side-decision algorithm (to be specified later), we can determine the region Q of the cutting that contains the solution to (P2). Because Q is the region of a $1/t$-cutting of Γ, we know that at most $|\Gamma|/t$ planes of Γ intersect Q. In particular, at most $|\Gamma|/t$ plane-sets intersect Q and hence, at least $(t-1)|\Gamma|/t$ plane-sets do not intersect Q. Since $|\Gamma| \geq m/2$, at least $(t-1)m/2t$ plane-sets do not intersect Q.

Let $(\triangle_i, \triangle_j)$ be a pair such that its plane-set does not intersect Q. Let Q' be the projection of Q on the plane Z_0. Because the plane-set of this pair does not intersect Q, we know that Q' intersects neither the boundary of $\triangle_i$ nor that of $\triangle_j$. Two cases arise:

Case 1. If either $\triangle_i$ or $\triangle_j$ does not intersect Q', then we know that their apex function is redundant and we can drop the constraint associated with this apexed triangle.

Case 2. If $Q' \subset \triangle_i \cap \triangle_j$, then we need to decide which constraint to drop. To this end, we consider the separating plane $\gamma_{i,j}$. Notice that inside the vertical extrusion of $\triangle_i \cap \triangle_j$ (and hence in Q), the plane $\gamma_{i,j}$ has the property that if we know which side of it contains the solution, then one of the constraints can be dropped. Since $\gamma_{i,j}$ does not intersect Q as $\gamma_{i,j}$ belongs to the plane-set of $(\triangle_i, \triangle_j)$, we can decide which side of $\gamma_{i,j}$ contains the solution to (P2) and drop one of the constraints.

Regardless of the case, if the plane-set of a pair $(\triangle_i, \triangle_j)$ does not intersect Q, then we can drop one of its constraints. Since at least $(t-1)m/2t$ plane-sets do not intersect Q, we can drop at least $(t-1)m/2t$ constraints. By choosing $t = 2$, we are able to drop at least $(t-1)m/2t = m/4$ constraints. Consequently, after $O(m)$ time, we are able to drop $m/4$ apexed triangles. By repeating this process recursively, we end up with a constant size problem in which we can compute the upper envelope of the functions explicitly and find the solution to (P2) using exhaustive search. Thus, the running time of this algorithm is bounded by the recurrence $T(m) = T(3m/4) + O(m)$ which solves to $O(m)$. Because $m = O(n)$, we can find the solution to (P2) in $O(n)$ time.

It remains to describe the side-decision algorithm. Given a plane γ, we want to decide in which side of γ lies the solution to (P2). To this end, we solve (P2) restricted to γ, i.e., with the additional constraint $(x, r) \in \gamma$. This approach was used by Megiddo [19], the idea is to recurse by reducing the dimension of the problem. Another approach is to use a slight modification of the chord-oracle described by Pollack et al. [23, Section 3].

Once the solution to (P2) restricted to γ is known, we can follow the same idea used by Megiddo [19] to find the side of γ containing the global solution to (P2). That is, we find the apex functions that define the minimum restricted to γ. Since $\phi(x) = F_P(x)$ is locally defined by these functions, we can decide in which side the minimum lies using convexity. We obtain the following result.

▶ **Lemma 17.** *Let R' be a convex trapezoid contained in P such that R' contains the geodesic center of P. Given the set of all apexed triangles of τ that intersect R', we can compute the geodesic center of P in $O(n)$ time.*

The following theorem summarizes the result presented in this paper.

▶ **Theorem 18.** *We can compute the geodesic center of any simple polygon P of n vertices in $O(n)$ time.*

References

1 Hee-Kap Ahn, Luis Barba, Prosenjit Bose, Jean-Lou De Carufel, Matias Korman, and Eunjin Oh. A linear-time algorithm for the geodesic center of a simple polygon. *CoRR*, abs/1501.00561, 2015.

2 Boris Aronov, Steven Fortune, and Gordon Wilfong. The furthest-site geodesic Voronoi diagram. *Discrete & Computational Geometry*, 9(1):217–255, 1993.

3 T. Asano and G.T. Toussaint. Computing the geodesic center of a simple polygon. Technical Report SOCS-85.32, McGill University, 1985.

4 Sang Won Bae, Matias Korman, and Yoshio Okamoto. The geodesic diameter of polygonal domains. *Discrete & Computational Geometry*, 50(2):306–329, 2013.

5 Sang Won Bae, Matias Korman, and Yoshio Okamoto. Computing the geodesic centers of a polygonal domain. In *Proceedings of CCCG*, 2014.

6 Sang Won Bae, Matias Korman, Yoshio Okamoto, and Haitao Wang. Computing the L_1 geodesic diameter and center of a simple polygon in linear time. In *Proceedings of LATIN*, pages 120–131, 2014.

7 Bernard Chazelle. A theorem on polygon cutting with applications. In *Proceedings of FOCS*, pages 339–349, 1982.

8 Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6(1):485–524, 1991.

9 H.N. Djidjev, A. Lingas, and J.-R. Sack. An $O(n \log n)$ algorithm for computing the link center of a simple polygon. *Discrete & Computational Geometry*, 8:131–152, 1992.

10 Herbert Edelsbrunner and Ernst Peter Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. on Graphics*, 9(1):66–104, 1990.

11 Leonidas Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert E Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1-4):209–233, 1987.

12 Leonidas J Guibas and John Hershberger. Optimal shortest path queries in a simple polygon. *Journal of computer and system sciences*, 39(2):126–152, 1989.

13 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

14 John Hershberger and Subhash Suri. Matrix searching with the shortest-path metric. *SIAM Journal on Computing*, 26(6):1612–1634, 1997.

15 Y. Ke. An efficient algorithm for link-distance problems. In *Proceedings of SoCG*, pages 69–78, 1989.

16 Der-Tsai Lee and Franco P Preparata. Euclidean shortest paths in the presence of rectilinear barriers. *Networks*, 14(3):393–410, 1984.

17 Jiří Matoušek. Approximations and optimal geometric divide-and-conquer. *Journal of Computer and System Sciences*, 50(2):203–208, 1995.

18 Jiří Matoušek. Construction of epsilon nets. In *Proc. of SoCG*, pages 1–10. ACM, 1989.

19 Nimrod Megiddo. On the ball spanned by balls. *Discrete & Computational Geometry*, 4(1):605–610, 1989.

20 J. S. B. Mitchell. Geometric shortest paths and network optimization. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 633–701. Elsevier, 2000.

21 B.J. Nilsson and S. Schuierer. Computing the rectilinear link diameter of a polygon. In *Proceedings of CG*, pages 203–215, 1991.

22 B.J. Nilsson and S. Schuierer. An optimal algorithm for the rectilinear link center of a rectilinear polygon. *Computational Geometry: Theory and Applications*, 6:169–194, 1996.

23 Richard Pollack, Micha Sharir, and Günter Rote. Computing the geodesic center of a simple polygon. *Discrete & Computational Geometry*, 4(1):611–626, 1989.

24 S. Suri. *Minimum Link Paths in Polygons and Related Problems.* PhD thesis, Johns Hopkins Univ., 1987.

25 Subhash Suri. Computing geodesic furthest neighbors in simple polygons. *Journal of Computer and System Sciences*, 39(2):220–235, 1989.