Available online at www.sciencedirect.com
ScienceDirect

Theoretical Computer Science 389 (2007) 1–11

Theoretical Computer Science
www.elsevier.com/locate/tcs

Query-point visibility constrained shortest paths in simple polygons[☆]

Ramtin Khosravi[*], Mohammad Ghodsi

Department of Computer Engineering, Sharif University of Technology, P.O. Box 11365-9517, Tehran, Iran
IPM School of Computer Science, P.O. Box: 19395-5746, Tehran, Iran

Received 2 June 2005; received in revised form 9 March 2007; accepted 1 July 2007

Communicated by S. Sen

Abstract

In this paper, we study the problem of finding the shortest path between two points inside a simple polygon such that there is at least one point on the path from which a query point is visible. We provide an algorithm which preprocesses the input in $O(n^2+nK)$ time and space and provides logarithmic query time. The input polygon has n vertices and K is a parameter dependent on the input polygon which is $O(n^2)$ in the worst case but is much smaller for most polygons. The preprocessing algorithm sweeps an angular interval around every reflex vertex of the polygon to store the optimal contact points between the shortest paths and the windows separating the visibility polygons of the query points from the source and the destination.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Computational geometry; Shortest path; Visibility

1. Introduction

In a number of applications, it is necessary for a moving object to have direct visibility from some viewpoint during its motion [1]. Examples are point-to-point communication, military applications, and moving guards. This imposes a constraint called *visibility constraint* on the well-known problem of finding the shortest path between two points. In the shortest path problem with single-point visibility constraint, the goal is to find the shortest path between two points such that there is at least one point on the path from which a given viewpoint is visible. This problem has been studied in [14] for various geometric domains including simple polygons, polygons with holes, and polyhedral surfaces. A more general constraint is studied in [13] where the path is required to meet a target polygon (not just a visibility polygon) where an $O(n)$ algorithm is given for the problem. The same problem is solved in [12] for the case of polygonal domains resulting in an $O(n \log n)$ algorithm. Extending the problem to visit multiple regions relates the problem to the well-known problems of TSP with neighborhoods [5,6], watchman route problem [2,20], zookeeper's problem [3,20], and safari route problem [19,21]. Dror et al. [4] have presented an algorithm for the problem of finding

[☆] This work was supported by a grant from the IPM School of CS (No. CS1382-2-02).

[*] Corresponding address: ECE Department, University of Tehran, P.O. Box 14395-515, Tehran, Iran. Tel.: +98 21 6111 4918.
E-mail addresses: rkhosravi@ece.ut.ac.ir (R. Khosravi), ghodsi@sharif.edu (M. Ghodsi).

0304-3975/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2007.07.003

the shortest path that visits k given convex polygons in a pre-specified order. Also, they have shown that the problem is NP-hard for the case of non-convex polygons.

In this paper, we study the *query-point* version of the visibility constraint, that is, finding the shortest path between two given points s and t inside a simple polygon P with n vertices, such that there is at least one point on the path from which a given query point q is visible. The given algorithm preprocesses the input in $O(n^2+nK)$ time, building a data structure of the same size that answers the queries in logarithmic time and space where K is a parameter dependent on the input polygon which is $O(n^2)$ in the worst case, but much smaller for most input polygons.

A simpler form of the problem is studied in [15] in which the goal is to find the shortest path from s to view q (without going to a destination point). The given algorithm takes a similar approach to the one taken in this paper, that is preprocessing the input using a radial sweep process around reflex vertices of P. The preprocess takes $O(n^2)$ time and space and is capable of answering the queries in $O(\log n)$ time. The result is improved by Knauer and Rote to $O(n \log n)$ preprocessing time and $O(n)$ space achieving the same query time [17]. Their approach is different from that of [15] and uses data structures for ray shooting queries [10] and lowest common ancestor queries [9] to compute and search the funnel structure made by the shortest path tree of s.

The main idea of the algorithm given in this work is to consider the segments inside P that can be considered as a window of the visibility polygon of a possible query point (a window is an edge of a visibility polygon which is not part of the boundary of the input polygon). If such a window separates both s and t from the visibility polygon of the query point, then the optimal path must touch a point on that window and reflect back to the destination. The problem is to find the optimal contact point between the optimal path and the window. To answer the queries efficiently, we preprocess the input using a radial sweep around each reflex vertex of the polygon to partition the windows around the reflex vertex to a number of sets such that knowing the set the query window belongs to, one can compute the optimal contact point in constant time. To find the desired partition, we study the behavior of the optimal contact point over a rotating sweep window.

The problem will be stated more elaborately in Section 2. The algorithm relies on the notion of funnel defined in [7]. In Section 3 we study some properties of the funnel related to the problem under considerations. Finally, the preprocessing algorithm which uses a radial sweep is presented and analyzed in Section 4.

2. Problem definition

We use the following notation throughout the paper:

- V_x: the visibility polygon of a point $x \in P$
- $\pi(x, y)$: the shortest path between two points x and y inside P
- $d(x, y)$: the length of $\pi(x, y)$
- $|xy|$: the length of the segment xy
- SPT(x): the shortest path tree from the source x [7]
- SPM(x): the shortest path map of P with source x [7].

In this problem, two points s and t are given inside a simple polygon P. The goal is to preprocess the input to answer queries of this type: given a query point $q \in P$, find the shortest path between s and t in P, such that there exists at least one point on the path from which q is visible. In other words, the shortest path between s and t that should have non-empty intersection with V_q. We call such a path, a *q-visible* path. Note that since P is a simple polygon, it can be verified that a q-visible path is unique [13]. Removing V_q from P yields in a number of disconnected regions we call *invisible regions*. Each invisible region has exactly one edge in common with V_q, called a *window*.

Since the query can be answered in $O(n)$ time without the preprocessing [13], our goal is to find a logarithmic query time. Since the complexity of the output path may be $O(n)$, we define two types of queries: one to find out the shortest distance, and another to report the shortest path. We answer the first type of queries in logarithmic time, and the second type in $O(L + \log n)$, where L is the complexity of the optimal path. Below, we consider the first type of query and will provide comments on the second type when necessary.

The following cases may occur for a query point q:

(1) At least one of s and t is visible from q. In this case, the shortest path between s and t is the answer.
(2) The points s and t are in two distinct invisible regions (i.e., are separated from V_q by two distinct windows). Again in this case, the shortest path between s and t is the answer.

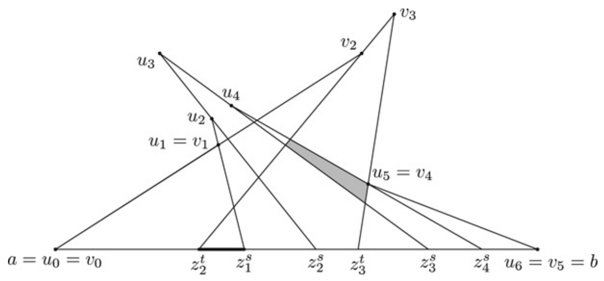

Fig. 1. Two funnels over the window ab. The segment $z_2^t z_1^s$ is I_{13}. The shaded part is $Cell_{43}$.

(3) Both s and t are in the same invisible region separated from V_q by a window w. In this case, the q-visible path from s to t touches w in exactly one point $c(w)$ [13]. (This comes from the uniqueness of shortest paths in simple polygons). We call this point the *optimal contact point* of the window w. Obviously, the two sub-paths from s to $c(w)$ and from $c(w)$ to t are regular shortest paths in P. So, $c(w)$ is the point on w such that $d(s, c(w))+d(t, c(w))$ is minimum.

Detecting whether any of the first two cases has been occurred is easy and can be done by finding the window separating s and t from V_q. The window separating s (resp. t) from V_q is specified by the last vertex of the shortest path from s (resp. t) to q. Thus, having computed the shortest path maps during the preprocessing phase, we can find the window w in $O(\log n)$ time using a standard point-location algorithm [16]. In the first two cases, we just have to report the shortest Euclidean distance from s to q, which is possible in $O(\log n)$ time provided that the SPM of s has been computed. The path itself can be reported in additional $O(L)$ time where L is the complexity of the path.

In the third case, the goal is to find $c(w)$ on the window w separating both s and t from V_q. This task is more complicated, and will be discussed in the rest of the paper. Section 3 shows how to find $c(w)$ on a given window w. Then in Section 4 we show how to preprocess the input to efficiently handle the query case, i.e., where w is computed upon receiving the query point q.

3. Characterizing the optimal contact point

In this section, we show how the optimal contact point $c(w)$ can be characterized on a given window w of P. Recall that $c(w)$ is the point on w such that $d(s, c(w)) + d(t, c(w))$ is minimum. The method is to partition w into intervals, such that the combinatorial structure of the shortest paths from s and t to all points on an interval is the same. This way, we can find the point on each interval that has the minimum total distance to s and t. Among all these points, $c(w)$ is the one with minimum total distance. To find the mentioned partition, we use the notion of *funnel* [7] which is the basic structure in study of the shortest paths in simple polygons.

Assume a and b to be the endpoints of a window w. We define two funnels over w with respect to s and t. We first give the definitions corresponding to the point s. Define the funnel $F_s(w)$ as $\pi(r, a) \cup \pi(r, b)$ where r is the last vertex common between the two paths $\pi(s, a)$ and $\pi(s, b)$ when considered from s to a and b respectively. We assume that the vertices on this funnel are named $a = u_0, u_1, \ldots, u_{m_s}, u_{m_s+1} = b$ in the ordered traversal from a to b. The region enclosed between $F_s(w)$ and w can be decomposed into triangles by extending the edges of $F_s(w)$ to intersect w (Fig. 1). Assume the extension of the edge $u_i u_{i+1} (0 \le i \le m_s)$ intersects w in z_i^s (hence, $z_0^s = a$ and $z_{m_s+1}^s = b$). The funnel $F_t(w)$ is defined similarly and is assumed to have the sequence of vertices $a = v_0, v_1, \ldots, v_{m_t+1} = b$ and the extension of the edge $v_j v_{j+1}$ $(0 \le j \le m_t)$ intersects w in z_j^t. We drop the parameter w from $F_s(w)$ and $F_t(w)$ whenever the window over which the funnels are defined is clear from the context. Since there are many symbols in this paper having a window as parameter, we omit the parameter from those symbols too when it can be clearly understood.

The set of segments $\{z_{i-1}^s z_i^s\}$ partition w into intervals with respect to the combinatorial structure of the shortest paths from s to the points on w. More precisely, the shortest path from s to any point on a segment $z_{i-1}^s z_i^s$ passes through u_i as the last vertex. The same property holds for the set of segments $\{z_{j-1}^t z_j^t\}$ with respect to t. Now consider the set of intervals on w obtained by overlapping these two sets of segments. The intersection of $z_{i-1}^s z_i^s$ and $z_{j-1}^t z_j^t$ defines an interval $I_{ij}(w)$ such that for any point $x \in I_{ij}(w)$, the last vertex of $\pi(s, x)$ is u_i and the last vertex of $\pi(t, x)$ is v_j.

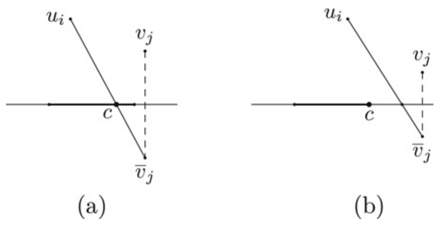

Fig. 2. Computing $c_{ij}(w)$ — the bold segment is I_{ij} and Ref_{ij} is the intersection point between $u_i\overline{v}_j$ and the horizontal line. The point labeled by c is c_{ij}. In case (a), c_{ij} is the same as Ref_{ij} while in case (b), c_{ij} is the right end of I_{ij} which is closer to Ref_{ij}.

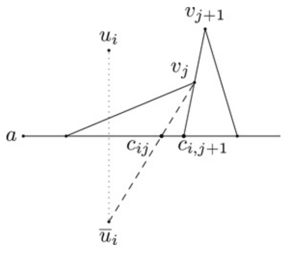

Fig. 3. Characterizing $c(w)$ — The point c_{ij} is the same as Ref_{ij} while $c_{i,j+1}$ is the left endpoint of $I_{i,j+1}$. In this case, $c(w) = c_{ij}(w)$.

Define $c_{ij}(w)$ as the point on $I_{ij}(w)$ with minimum total distance to s and t. Since all points on I_{ij} has u_i and v_j as the last vertices along shortest paths from s and t respectively, c_{ij} is the point on I_{ij} such that $|u_i c_{ij}| + |c_{ij} v_j|$ is minimum. To find c_{ij}, let ℓ be the line supporting w, and $\overline{v}_j$ be v_j reflected about ℓ. Define $Ref_{ij}(w)$ as the intersection of the segment $u_i\overline{v}_j$ and ℓ. If Ref_{ij} lies inside I_{ij}, then c_{ij} is the same as Ref_{ij}, otherwise, it is the endpoint of I_{ij} which is closer to Ref_{ij} (Fig. 2).

Among the set of points $\{c_{ij}(w)|$ for all intervals $I_{ij}(w)\}$ one has the minimum total distance to s and t which is the optimal contact point $c(w)$. The following lemma uses properties of funnels to show at most one interval satisfies the property $c_{ij} = Ref_{ij}$.

Lemma 3.1. *For an arbitrary window $w = ab$, if there exists a pair of vertices (u_i, v_j) such that $c_{ij} = Ref_{ij} \in I_{ij}$, then for any interval $I_{i'j'} \neq I_{ij}$, $Ref_{i'j'} \notin I_{i'j'}$. Furthermore, if $I_{i'j'}$ is closer to (farther from) a than I_{ij}, then $c_{i'j'}$ is the endpoint of $I_{i'j'}$ farther from (closer to) a.*

Proof. Assume I_{ij} is not the farthest interval from a and consider the one of its two adjacent intervals which is farther from a. This interval is either $I_{i,j+1}$ or $I_{i+1,j}$. For the first case, let $\overline{u}_i$ be u_i reflected about w and ℓ be the line passing through v_j and v_{j+1}. We name the half-plane defined by ℓ containing a (resp. b) as H_a (resp. H_b). It easy to see that $\overline{u}_i \in H_a$ (because $c_{ij} \in I_{ij}$ is in H_a, by assumption). Since $v_{j+1} \in \ell$, $Ref_{i,j+1}$ is in H_a, so $c_{i,j+1}$ is the endpoint of $I_{i,j+1}$ closer to a (which is z_j^t in this case). This property holds for $I_{i+1,j}$ interval too. So, the interval adjacent to I_{ij} and farther from a has its optimal point at its endpoint closer to a. Similar argument yields to the property for the interval adjacent to I_{ij} and closer to a that has its optimal point at its endpoint farther from a.

Now consider an interval $I_{i'j'}$ with its endpoint closer to a as the optimal point. Without loss of generality, assume $I_{i',j'+1}$ be its neighbor farther from a. Similar argument as above can be used to verify that $I_{i',j'+1}$ has its optimal point in its endpoint closer to a too. Using induction, we can conclude that any interval farther from I_{ij} has its endpoint closer to a as its optimal point. Same argument can be used to conclude that any interval closer to I_{ij} has its endpoint farther from a as its optimal point. This completes the proof of the lemma. □

Assume I_{ij} has the property that $c_{ij} = Ref_{ij}$ and without loss of generality, assume $I_{i,j+1}$ is its adjacent interval farther from a (Fig. 3). Furthermore, assume v_{j+1} is the parent of v_j in SPT(t). The total distance from c_{ij} to s and t is $D = d(s, u_i) + |u_i c_{ij}| + d(t, v_j) + |v_j c_{ij}|$. The same parameter for $c_{i,j+1}$ is $D' = d(s, u_i) + |u_i c_{i,j+1}| + d(t, v_{j+1}) + |v_{j+1} c_{i,j+1}|$. Since v_{j+1} is the parent of v_j, we have $d(t, v_j) = d(t, v_{j+1}) + |v_j v_{j+1}|$, so $D = d(s, u_i) + |u_i c_{ij}| + d(t, v_{j+1}) + |v_j v_{j+1}| + |v_j c_{ij}|$. On the other hand, $|v_{j+1} c_{i,j+1}| = |v_j v_{j+1}| + |v_j c_{i,j+1}|$, so $D' = d(s, u_i) + |u_i c_{i,j+1}| + d(t, v_{j+1}) + |v_j v_{j+1}| + |v_j c_{i,j+1}|$. Ignoring the common terms in D and D', we have $D < D'$, because $|u_i c_{ij}| + |v_j c_{ij}| < |u_i c_{i,j+1}| + |v_j c_{i,j+1}|$ (which is true since $c_{ij} = Ref_{ij}$). If we assume v_j is a

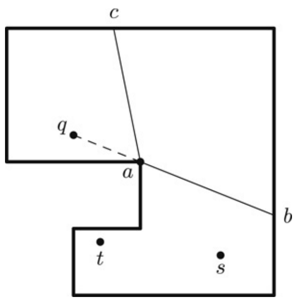

Fig. 4. The segment ab is a separating window while ac is not.

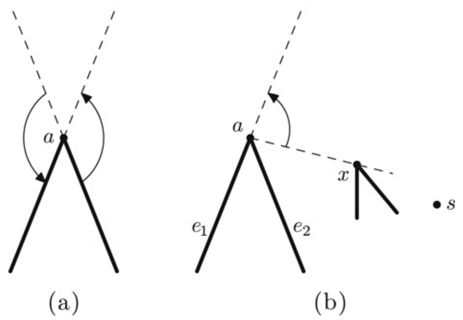

Fig. 5. (a) Two angular intervals defining windows around a reflex vertex a. (b) The single interval defining separating windows with respect to s around a where x is the last vertex on $\pi(s, a)$.

parent of v_{j+1}, similar arguments leads to the same result. The fact that $c_{i,j+1}$ lies on $v_j v_{j+1}$ is the key property used in these arguments. Similar arguments and the induction used in the proof of Lemma 3.1 can be used together to prove the following lemma which will be used in determining interval events during preprocessing.

Lemma 3.2. *For an arbitrary window $w = ab$, if there exists a pair of vertices (u_i, v_j) such that $c_{ij} = Ref_{ij} \in I_{ij}$, then c_{ij} is the optimal contact point $c(w)$. If no such pair exists, then $c(w)$ is either a or b.*

4. The preprocessing algorithm

To answer the queries efficiently, we must pre-compute the optimal contact point on all segments in P that can be a window of a query point, so that upon receiving a query point q, we find the window separating s and t from q and report the optimal contact point efficiently. We refer to such a segment of P as a *separating window*. Informally, a separating window is a window of the visibility polygon of an arbitrary point x that separates both s and t from V_x. For example, in Fig. 4 the segment ab is a separating window, since there exists a point (like q) invisible from s and t, such that ab is a window of V_q that separates it from s and t, but there is no such point for the segment ac. So, ac is not a separating window.

To simplify the specification of the set of separating windows, we first define the set of all windows separating s from possible query points and then generalize the definition to cover both s and t. To specify the set of all windows separating s from possible query points, we consider each reflex vertex of P and find the set of separating windows having that vertex as an endpoint. Assume a is a reflex vertex of P. Considering all possible query points inside P, we may have a set of windows associated with a that are defined by the rays emanating from a in two angular intervals between the extension of each edge incident to a and the other edge (Fig. 5(a)). Not all the windows defined by the two intervals mentioned are separating. To restrict the set to separating windows, consider x as the last internal vertex on the shortest path $\pi(s, a)$. If x lies outside both angular intervals, then there is no separating window around a. Otherwise, assume that it lies inside the interval defined by e_2 and the extension of e_1 where e_1 and e_2 are the edges incident to a (Fig. 5(b)). The angular interval defining the set of separating windows around a is bounded by the extension of e_1 and the ray emanating from a passing through x. We denote this set of separating windows by $Sep_s(a)$. Now it is easy to see that the set of windows separating both s and t from possible query points is $Sep(a) = Sep_s(a) \cap Sep_t(a)$. Since both $Sep_s(a)$ and $Sep_t(a)$ are angular intervals, so is $Sep(a)$.

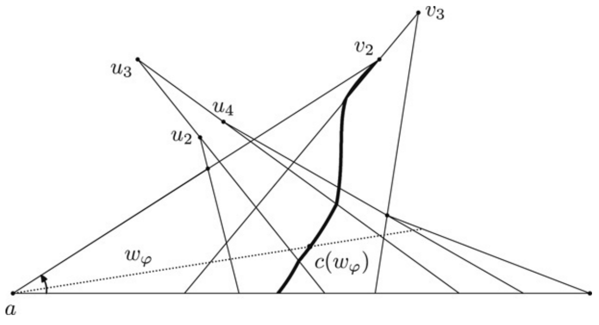

Fig. 6. The location of the optimal contact point $c(w_\varphi)$ is shown by the bold curve as w_φ (dotted line) sweeps $Sep(a)$.

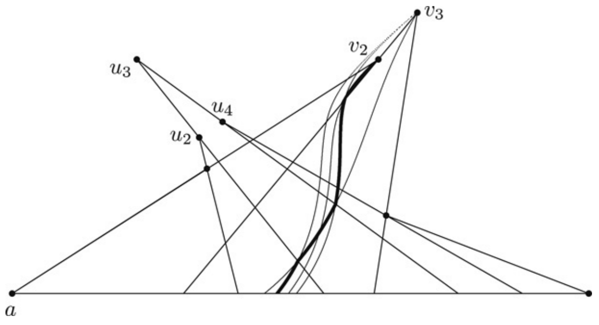

Fig. 7. The curve $\mathcal{C}^*$, shown as the bold curve is made up of parts of $\mathcal{C}_{ij}$ curves. Here, those curves contributed to $\mathcal{C}^*$ are $\mathcal{C}_{23}$, $\mathcal{C}_{33}$, $\mathcal{C}_{43}$, and $\mathcal{C}_{42}$ in that order.

For a reflex vertex a, we partition $Sep(a)$ into a number of sets such that for every window w in a set, the interval containing $c(w)$ is the same, so the pair of last vertices on $\pi(s, c(w))$ and $\pi(t, c(w))$ is the same. Upon receiving a query point, we first compute the corresponding window, then search the partition to find the set the window belongs to. Knowing the set, we can find the optimal contact point in constant time.

To partition $Sep(a)$, we use a radial sweep around a. There are two kinds of events in the sweep process: angles at which the optimal contact point changes its interval which we call *interval events*, and angles at which the structure of the funnel changes *funnel events*. We consider the two types of events subsequently.

4.1. Interval events

To compute and handle interval events, we need to study the behavior of the optimal contact point during a radial sweep of $Sep(a)$. Fig. 6 shows an example. The dotted line shows the sweep window w_φ at some instant during sweep. As w_φ rotates around a, the optimal contact point $c(w_\varphi)$ moves along the bold curve shown in the figure. To formally define the curve, let $Sep(a)$ be the angular interval $[\alpha, \alpha']$ and w_φ denote the window with endpoint a and angle φ ($\alpha \leq \varphi \leq \alpha'$). The optimal curve $\mathcal{C}^*$ is defined as the set $\{c(w_\varphi) | \alpha \leq \varphi \leq \alpha'\}$. As can be seen in the figure, $\mathcal{C}^*$ has some changes when crossing extension segments of the funnels. We will see how these intersections define the interval events.

Each pair of vertices (u_i, v_j) defines a curve tracing $Ref_{ij}(w_\varphi)$ along the sweep. As we will see shortly, $\mathcal{C}^*$ is made up of parts of these curves. More formally, consider a pair of vertices (u_i, v_j) from the two funnels $F_s(w_\alpha)$ and $F_t(w_\alpha)$ respectively. Let β be the smallest of angles made by au_i and av_j segments and w_α. Define $\mathcal{C}_{ij}$ as the set of points $\{Ref_{ij}(w_\varphi) | \alpha \leq \varphi \leq \beta\}$. This set is a simple, continuous curve leading to the vertex making the angle β.

Now consider the sweep window w_φ, and assume the optimal contact point c lies on the interval I_{ij} (i.e., $c = c_{ij}$). From Lemma 3.2 we conclude that c is the same point as Ref_{ij}, so it lies on $\mathcal{C}_{ij}$. To define the condition of c lying on I_{ij} more precisely, we define $Cell_{ij}$ as the intersection of the triangles $\triangle z^s_{i-1} u_i z^s_i$ and $\triangle z^t_{j-1} v_i z^t_i$ (Fig. 1). So, parts of $\mathcal{C}_{ij}$ which are inside $Cell_{ij}$ are parts of $\mathcal{C}^*$ (Fig. 7).

The following lemma is used to show the continuous behavior of $\mathcal{C}^*$ at intersections with extension segments.

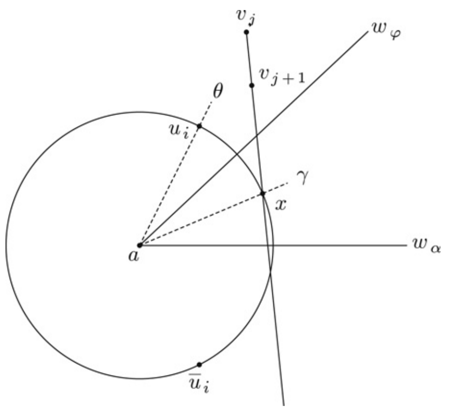

Fig. 8. Determining interval events — there is an intersection between $\mathcal{C}_{ij}$ and the edge of $Cell_{ij}$ bounded by $v_j v_{j+1}$ when the sweep window reaches w_φ.

Lemma 4.1. *Let $Cell_{ij}$ and $Cell_{i'j'}$ be two adjacent cells. If $\mathcal{C}_{ij}$ intersect the common boundary of the two cells at x, then either x is a vertex of P, or x belongs to $\mathcal{C}_{i'j'}$ too.*

Proof. To be adjacent to $Cell_{ij}$, the pair (i', j') must be one of $(i, j+1)$, $(i, j-1)$, $(i+1, j)$, $(i-1, j)$. Consider the first case, $Cell_{i,j+1}$. The common boundary lies on the line ℓ supporting $v_j v_{j+1}$. Since $x \in \mathcal{C}_{ij}$, it lies on the segment $\overline{u}_i v_j$ where $\overline{u}_i$ is the reflection of u_i about ax (a is the fixed endpoint of the sweep window). The fact that x lies on $\overline{u}_i v_j$ and ℓ at the same time leaves two possibilities: either $x = v_j$ or $\overline{u}_i \in \ell$. The first case implies x is a vertex of P. In the second case, x is the intersection of $\overline{u}_i v_{j+1}$ and the sweep window, so it belongs to $\mathcal{C}_{i,j+1}$. The same argument applies to other cases for (i', j'). □

Based on the preceding discussions, the following lemma determines the structure of $\mathcal{C}^*$.

Lemma 4.2. *$\mathcal{C}^*$ is a continuous simple curve and is constructed from concatenation of parts of $\mathcal{C}_{ij}$ curves and segments which are parts of the edges of P.*

Proof. Since $\mathcal{C}^*$ is obtained from a radial sweep, it cannot intersect itself, so it is a simple curve. Suppose at some moment during the sweep process, the optimal contact point is in $Cell_{ij}$. We trace the curve $\mathcal{C}_{ij}$ along the sweep direction until we meet the boundary of the cell for the first time. Assume this happens at point x which is on the boundary common to $Cell_{i'j'}$. From Lemma 4.1, $\mathcal{C}_{i'j'}$ intersects $\mathcal{C}_{ij}$ at x. This way, if we start from $c(w_\alpha)$ and trace the $\mathcal{C}_{ij}$ curves, changing the curve when they go outside their corresponding cell, we will have the curve $\mathcal{C}^*$ except for those parts in which a curve goes outside the polygon P (this is related to the second case in Lemma 3.2). In those parts, the curve is substituted with the relevant portion of the boundary of P. This way, the continuity of $\mathcal{C}^*$ comes directly from the continuity of $\mathcal{C}_{ij}$ curves. □

To compute the interval events we start from the initial interval containing $c(w_\alpha)$. When $\mathcal{C}^*$ enters a cell $Cell_{ij}$, we compute the edge of that cell from which the curve $\mathcal{C}_{ij}$ exits. This creates the next event at which $\mathcal{C}^*$ enters the cell adjacent to $Cell_{ij}$ along that edge. This processed is repeated till the whole interval of $Sep(a)$ is processed. Note that there may be funnel events (discussed later) that are to be considered during radial sweep.

To find the edge from which the current curve (say $\mathcal{C}_{ij}$) exits, we must consider each edge of $Cell_{ij}$ in turn. Without loss of generality, assume the edge corresponding to the extension edge of $v_j v_{j+1}$ is considered. Define ℓ as the line passing through v_j and v_{j+1}, and $Circ_i$ as the circle centered at a and passing through u_i (Fig. 8). As the sweep window rotates around a, $\overline{u}_i$ (u_i reflected about the sweep window) moves along $Circ_i$ towards u_i. As stated in the proof of Lemma 4.1, to have an intersection between $\mathcal{C}_{ij}$ and ℓ, $\overline{u}_i$ must be on ℓ. This happens at intersections of ℓ and the part of $Circ_i$ between u_i and the initial position of $\overline{u}_i$. Let x be such an intersection point and θ (resp. γ) be the angle between au_i (resp. ax) and the current sweep window. Then the event corresponding to the intersection point x occurs at angle $\frac{1}{2}(\theta + \gamma)$. Since there are at most four edges bounding a cell, this computation takes constant time for each event. Finally, we must select the event with minimum sweep angle and discard the others.

The following lemma bounds the number of interval events.

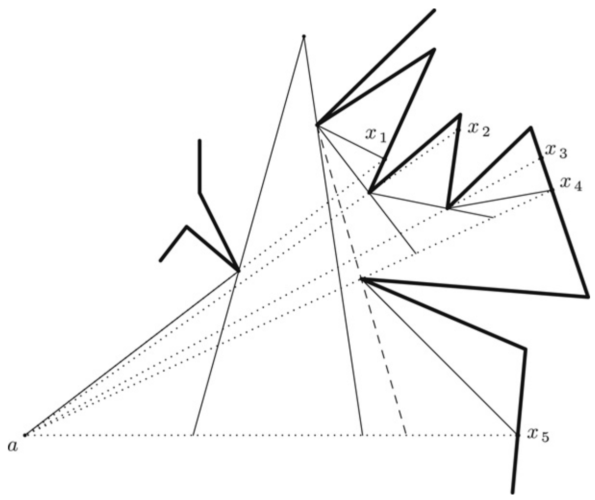

Fig. 9. Funnel events for one of the funnels. The angular interval between ax_1 and ax_5 defines the set of all separating windows. The sweep starts from ax_1. Thick solid lines are the edges of P, thin solid lines are the extension segments, dotted lines show the swap window in different positions, and the dashed line is an extension segment added after visiting ax_4.

Lemma 4.3. *The number of interval events is $O(n^2)$ per reflex vertex.*

Proof. To provide a bound on the number of interval events, we must count the number of times $\mathcal{C}^*$ crosses the extension segments. Since $\mathcal{C}^*$ is made up of parts of $\mathcal{C}_{ij}$ curves, we first check the number of times each such curve can cross the extension segments. This is easy to check based on the method discussed to determine the interval events, since the line ℓ intersects $Circ_i$ in at most two points. So, each $\mathcal{C}_{ij}$ curve has a constant number of corresponding events. Since there are $O(n^2)$ curves, the total number of interval events for a reflex vertex is $O(n^2)$. □

There are examples showing this bound is tight. Unfortunately they are hard to illustrate, so we show how to construct such a case in Appendix. In most practical cases, the number of such events is smaller than this and is close to n. So, we define a new parameter K which is the maximum number of interval events per reflex vertex in P and state the running time and space of the algorithm in terms of K. We expect K to be $O(n)$ for most polygons. Note that in the worst case, it is possible to have a polygon with $\theta(n)$ reflex vertices each with $\theta(n^2)$ interval events. The method presented in Appendix can be used to construct this example too.

4.2. Funnel events

To define the funnel events, we consider each funnel individually. This is possible since the changes in the funnels occur independently. We start with the funnel events corresponding to F_s. Observe that the set of separating windows ($Sep_s(a)$) is a subset of the visibility polygon of a, V_a bounded to w_α and $w_{\alpha'}$. It is easy to see that the moments at which the sweep window passes through the vertices of V_a, the structure of the funnel changes. So, for any reflex vertex, there are at most $O(n)$ funnel events. The problem is to update the funnel efficiently at these moments. We assume that SPT(s) is computed previously as well SPM(s) and V_a such that we may traverse the vertices of V_a in order. The key to efficient update is to start the sweep process from the separating window closest to s. According to our notation, either w_α or $w_{\alpha'}$ has this property. Here, we assume w_α is the one.

Initially, we compute the funnel over w_α. As the sweep window rotates around a, we may encounter a new vertex from V_a, such as p. If p is not a reflex vertex of P, the effect of this event is only the change in the edge of P on which the non-fixed endpoint of the sweep window moves. Otherwise, we have encountered a new node in SPT(s) and this may cause a vertex to be added to the funnel. It is possible that the newly added vertex deletes parts of the funnel too. This happens when the parent of the added vertex in SPT(s) has another child belonging to the funnel right before the event. For example, in Fig. 9, the initial funnel is built over ax_1. New vertices are added to the funnel as the sweep window meets the points x_2 and x_3. At x_4, a new vertex is added with the dashed extension segment introduced and some vertices are deleted from the funnel.

Note that we must keep track of the last reflex vertex of V_a visited. Having this, we can compute the change that should be made to the funnel in constant time. This is possible if we store *parent* pointers in SPT(s). When some vertices are deleted, observe that the deleted vertices form a subtree of the SPT, so the deletion can be done in constant time too. Also, we must have the vertices of the initial funnel in sorted order. This is possible if we make the recursive calls made during the DFS traversal in the shortest path algorithm sorted in some fixed direction (e.g. clockwise). Since the funnel structure used in the algorithm is stored in a *finger search tree* [8], the list of vertices in the funnel can be arranged in sorted order in linear time.

Changes to the funnel F_t are computed exactly in a similar manner. The total number of funnel events is $O(n)$.

4.3. Analysis of the preprocessing

Summarizing the method described above, we take the following steps during the preprocessing phase:

(1) Compute SPM(s), SPT(s), SPM(t), and SPT(t)
(2) For each reflex vertex a do the following:
   (a) Compute $Sep(a)$.
   (b) Compute the portion of V_a bounded by $Sep(a)$.
   (c) Compute the initial funnels and the extension edges
   (d) Perform the radial sweep starting from the closest separating window to s and t.

The first step can be done in $O(n)$ time using the algorithm of [7]. Step (a) involves finding the last vertex on $\pi(s, a)$ and $\pi(t, a)$ which can be done in $O(\log n)$. The visibility computation in step (b) can be done using the linear time algorithm of [18,11]. The funnel and the extension edges in step (c) are derived directly from the SPMs in $O(n)$ time. Step (d) involves computing and handling both types of events which are $O(n + K)$ in total and needs constant time per event. After this step, the events are obtained in sorted angular order. This leads to $O(n + K)$ preprocess for each reflex vertex. The partition for the reflex vertex is of size $O(n + K)$ too. As there are $O(n)$ reflex vertices, the total preprocessing time is $O(n^2 + nK)$ and $O(n^2 + nK)$ space is needed to store the partitions.

4.4. Processing queries

Upon receiving a query, we find the windows separating s and t from the query point q in $O(\log n)$ time (by finding the cells of SPM(s) and SPM(t) containing q). Assuming the windows are the same (say $w = ab$), we perform a binary search on the partition associated with the reflex vertex a which requires $O(\log n)$ time. Associated with the partition is the pair of vertices (u_i, v_j) which are the last vertices on $\pi(s, c(w))$ and $\pi(t, c(w))$. So, finding the optimal point $c(w)$ takes constant time. If we need to report the shortest paths (not just the distances), we can find the shortest path from s to $c(w)$ appended by the path from $c(w)$ to t which is an easy task since both shortest path maps are computed in advance. So, we have our main result as the following:

Theorem 4.1. *Given a simple polygon P with n vertices, and two points s and t inside P, we can preprocess the input in $O(n^2 + nK)$ time and space so that we can find the length of the shortest path from s to t such that a query point q is visible from at least one point of the path in $O(\log n)$, where K is the maximum number of interval events per reflex vertex which is $O(n^2)$ in the worst case. The path itself can be reported with an additional cost of $O(L)$ where L is the complexity of the output path.*

5. Conclusion

We presented an algorithm to preprocess the input polygon in $O(n^2 + nK)$ time and space to answer the queries to find the shortest distance between two given points constrained to view a query point in logarithmic time, where K is a parameter dependent on the input polygon which is $O(n^2)$ in the worst case, but $O(n)$ for most input polygons. In particular, we studied the behavior of the optimal contact point over a rotating segment. An extension to this problem is to consider the query not just a point, but another geometric object like a segment. For a segment, the algorithm can still work considering the appropriate window from the weak visibility polygon of the segment. For more complex objects like a polygon, the challenge is to find the appropriate window from the visibility polygon. Once the window is computed, the rest of the algorithm is similar to that of a point. Another possible extension is to require the path

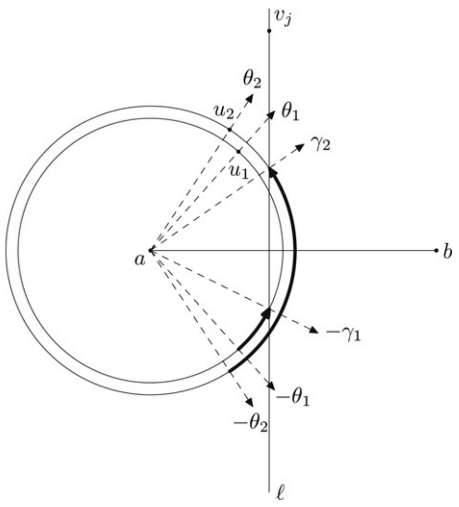

Fig. A.1. The construction for the first two points in U. The angles spanned by the heavy directed arcs indicate the amount of sweep required for the first and second interval events.

to visit a query polygon (not necessarily a visibility polygon). If we fix the shape of the query polygon and allow translation and rotations inside the input polygon, then we can use the results of this paper for tracking the optimal contact point on a rotating segment.

Appendix. Constructing funnels with $\Theta(n^2)$ interval events

In this appendix, we show how to construct two funnels F_s and F_t over a window $w = ab$ such that the number of interval events is $\Theta(n^2)$ where n is the total number of vertices in both funnels. Let a be the reflex vertex of the window over which the funnels are built. We assume a polar coordinate system with a as the origin. Without loss of generality, we may assume the polar angle of b is zero (this may be fixed using a rotation transformation).

We first show how to place the sequence of vertices $U = (u_1, u_2, \ldots, u_k)$ of the funnel F_s such that the optimal curve intersects an extension segment of F_t (like $v_j z_j^t$) in $\Theta(k)$ points. To start with a simple case, assume the extension segment $v_j z_j^t$ is perpendicular to w and ℓ is the line supporting the extension segment and both x and y coordinates of v_j are positive (Fig. A.1).

Imagine we have such a placement. So, the curves $\mathcal{C}_{1j}, \mathcal{C}_{2j}, \ldots, \mathcal{C}_{kj}$ intersect ℓ in a sequence of points ordered in decreasing distance to v_j (since they are formed by a radial sweep). To characterize the intersection points, observe that as the sweep window rotates around a, $\overline{u}_i$ (u_i reflected about the window) moves on an arc A_i of the circle $Circ_i$ centered at a and passing through u_i. A_i starts at angle $-\theta_i$ and spans an angular interval of size twice the total sweep interval. It can be easily checked that the intersections between ℓ and $\mathcal{C}_{ij}$ correspond to the intersections between ℓ and A_i which is at most two. We call an intersection between ℓ and $Circ_i$ a *low* intersection if it happens in negative y half-plane and a *high* intersection otherwise. Let γ_i be the polar angle of the high intersection of ℓ and $Circ_i$. The low intersection will have $-\gamma_i$ polar angle.

To keep the optimal curve continuous, the intersection points on ℓ must alternate between low and high types. Assuming the first intersection is low, it happens when the polar angle of $\overline{u}_1$ is $-\gamma_1$. This happens at sweep angle $\theta_1 - \gamma_1$. The sequence of intersections continues as the angle of $\overline{u}_2$ becomes γ_2, the angle of $\overline{u}_3$ becomes $-\gamma_3$ and so on. The sequence of sweep angles of the intersections will be $(\theta_1 - \gamma_1, \theta_2 + \gamma_2, \theta_3 - \gamma_3, \ldots)$. Since the sequence is in increasing order, we have:

$$\begin{aligned}
&\theta_2 - \theta_1 > -(\gamma_1 + \gamma_2)\\
&\theta_3 - \theta_2 > \gamma_2 + \gamma_3\\
&\theta_4 - \theta_3 > -(\gamma_3 + \gamma_4)\\
&\vdots
\end{aligned}$$

Assuming d is the distance from a to the line ℓ, choose r_1 as $d + \varepsilon_r$ where ε_r is a small positive number (we will provide comments on how small it should be). Let r_i be $r_{i-1} + \varepsilon_r$ for $1 < i \le k$. Set $\theta_1 = \gamma_1 + \varepsilon_\theta$. Again, ε_θ is a small positive number. To compute θ_i from θ_{i-1}, we face two constraints: one that is imposed by the structure of the funnel (outward convexity of the two parts of the boundary), and one imposed from the above inequalities. We add the minimum amount required to satisfy both constraints to θ_{i-1}. The only problem is that adding the amounts caused by the inequalities when computing θ_i for odd values of i, causes the computed angles to quickly increase until the y-coordinates of θ_i becomes less than θ_{i-1}. At this point the sequence fails to form the funnel shape. This can be solved by choosing ε_r small enough, making γ_i values small accordingly.

The sequence U computed above forms one part of the boundary of F_s. The other part is not necessary to create and we can connect u_k to b directly. To construct the funnel F_t, we put two vertices v_1 and v_2 with equal x coordinates such that the construction of the vertices in U can be done regarding ℓ as the line supporting $v_1 v_2$. Then we add the vertices $v_3, v_4, \ldots, v_l$ towards positive x coordinates such that all the extension segments $v_j z_j^t$ are as close enough to ℓ so every curve $\mathcal{C}_{1j}, \mathcal{C}_{2j}, \ldots, \mathcal{C}_{kj}$ intersects $v_j z_j^t$ in $\theta(k)$ points alternating between low and high types.

This way, the curves contributed to $\mathcal{C}^*$ will be $\mathcal{C}_{11}, \mathcal{C}_{12}, \ldots, \mathcal{C}_{1l}, \mathcal{C}_{21}, \ldots, \mathcal{C}_{kl}$. So, it is possible to have cases in which there are $\theta(n^2)$ interval events for a reflex vertex.

References

[1] D.Z. Chen, Developing algorithms and software for geometric path planning problems, ACM Comput. Surveys 28 (4es) (1996) 18.

[2] W.-P. Chin, S. Ntafos, Watchman routes in simple polygons, Discrete Comput. Geom. 6 (1) (1991) 9–31.

[3] W.-P. Chin, S. Ntafos, The zookeeper route problem, Inform. Sci. 63 (3) (1992) 245–259.

[4] M. Dror, A. Efrat, A. Lubiw, J.S.B. Mitchell, Touring a sequence of polygons, in: Proc. 35th ACM Sympos. Theory Comput., 2003.

[5] A. Dumitrescu, J.S.B. Mitchell, Approximation algorithms for TSP with neighborhoods in the plane, in: Symposium on Discrete Algorithms, 2001, pp. 38–46.

[6] J. Gudmundsson, C. Levcopoulos, A fast approximation algorithm for TSP with neighborhoods and red–blue separation, in: Lecture Notes in Computer Science, vol. 1627, 1999, pp. 473–482.

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

[8] L.J. Guibas, E. McCreight, M. Plass, J. Roberts, A new representation for linear lists, in: Proc. 9th Annu. ACM Sympos. Theory Comput., 1977, pp. 49–60.

[9] D. Harel, R.E. Tarjan, Fast algorithms for finding nearest common ancestors, SIAM J. Comput. 13 (2) (1984) 338–355.

[10] J. Hershberger, S. Suri, A pedestrian approach to ray shooting: Shoot a ray, take a walk, J. Algorithms 18 (1995) 403–431.

[11] B. Joe, R.B. Simpson, Correction to Lee's visibility polygon algorithm, BIT 27 (1987) 458–473.

[12] R. Khosravi, M. Ghodsi, Shortest paths in polygonal domains with polygon-meet constraints, in: Proc. 19th European Workshop Comput. Geom., 2003, pp. 137–142.

[13] R. Khosravi, M. Ghodsi, Shortest paths in simple polygons with polygon-meet constraints, Inform. Process. Lett. 91 (2004) 171–176.

[14] R. Khosravi, M. Ghodsi, Shortest paths with single-point visibility constraints, Sci. Iran. 13 (1) (2006) 25–32.

[15] R. Khosravi, M. Ghodsi, The fastest way to view a query point in simple polygons, in: Abstracts of the 21st European Workshop on Computational Geometry, 2005, pp. 187–190.

[16] D.G. Kirkpatrick, Optimal search in planar subdivisions, SIAM J. Comput. 12 (1) (1983) 28–35.

[17] C. Knauer, G. Rote, Shortest inspection-path queries in simple polygons. Technical Report B-05-05, Institut für Informatik, Freie Universität Berlin, 2005.

[18] D.T. Lee, Visibility of a simple polygon, Comput. Vision Graph. Image Proc. 22 (1983) 207–221.

[19] S. Ntafos, Watchman routes under limited visibility, Comput. Geom. Theory Appl. 1 (3) (1992) 149–170.

[20] X. Tan, Approximation algorithms for the watchman route and zookeeper's problems, Discrete Appl. Math. 136 (2–3) (2004) 363–376.

[21] X. Tan, T. Hirata, Finding shortest safari routes in simple polygons, Inform. Process. Lett. 87 (4) (2003) 179–186.