The Fastest Way to View a Query Point in Simple Polygons

Ramtin Khosravi* Mohammad Ghodsi†

Abstract

In this paper, we study the problem of finding the shortest path from a given source point in a simple polygon to some point visible from a given query point. We will present an algorithm based on the notion of funnels in simple polygons. The algorithm preprocesses the input containing a simple polygon and a source point to produce a data structure to answer the queries in logarithmic time. The time and space required for preprocessing is quadratic in size of the simple polygon.

1 Introduction

The shortest path problem is a well-known problem for the domain of simple polygons. The problem is studied with several variations. One of the these variations is to constrain the shortest path to view a query point from at least one point on the path [6, 5]. Another similar constraint is studied in [4] where the path is required to meet a target polygon where an $O(n)$ algorithm is given for the problem. In this paper, we study the problem of finding the shortest path must be taken from a given source point in a simple polygon to view a query point.

To define the problem more precisely, let P be a simple polygon with n vertices. Suppose a source point s is given in P. The goal is to preprocess the input to answer queries of this type: given a query point $q \in P$, find the shortest distance one needs to travel from s to see q. More precisely, we want to find a point c visible from q that has the shortest distance from s. Note that if the query point q is visible from s, the point c is the s itself, so throughout the paper, we assume the query point q is given somewhere outside the visibility polygon of s.

The query can be answered in $O(n)$ time without preprocessing [4], so our goal is to find a logarithmic query time. Since the complexity of the 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. Our goal is to answer the first type of queries in logarithmic time, and the second type in $O(k + \log n)$, where k is the length of the optimal path. In this paper, we consider the first type of query and will provide comments on the second type when necessary.

The algorithm relies on the notion of funnel defined in [1]. In section 2 we study some properties of the funnel related to the problem under considerations. Our algorithm, presented in section 3, is based on the fact that the desired path always ends on a window of the visibility polygon of q. So, during the preprocessing phase, we compute a partition of all possible windows of visibility polygons in P into sets such that knowing the set a window belongs to, we can answer the ending point of the path efficiently.

2 Basic Properties

We use the notation V_p for the visibility of a point $p \in P$, $\pi(x, y)$ for the shortest path between two points x and y inside P, SPT(s) for the shortest path tree from s, and SPM(s) for the shortest path map of P with respect to s. Removing V_q from P results 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 s is invisible from q, it lies in an invisible region. It is easy to see that the point c lies on the window w separating s from V_q. More precisely, c is the point on w that has the shortest distance to s (Fig. 1). From now on, we refer to such a point c as the *optimal point* $c(w)$ to show explicitly the window it belongs to.

We use the notion of *funnel* as defined in [1]. Assume a and b be the endpoints of w. Define the funnel $F(w)$ as $\pi(r, a) \cup \pi(r, b)$ where r is the deepest common ancestor of a and b in the last common vertex between the two paths $\pi(s, a)$ and $\pi(s, b)$ when considered from s to a and b (Fig. 2). We assume the vertices on the funnel are named $a = v_0, v_1, \ldots, v_k, v_{k+1} = b$ in the ordered traversal from a to b. The region enclosed between $F(w)$ and w can be decomposed into triangular regions by extending the edges of $F(w)$ to intersect w. Assume the extension of the edge $v_i v_{i+1} (0 \leq i \leq k)$ intersects w in x_i (hence, $x_0 = a$ and $x_k = b$). The shortest path from s to points on the segment $x_i x_{i+1}$ passes through v_i as the last vertex. Denote the sequence of angles between the extension edges and the window w by $(\theta_0, \theta_1, \ldots, \theta_k)$, such that $\theta_i = \angle b x_i v_i (0 \leq i \leq k-1)$ and θ_k is $180° - \angle a b v_k$.

*Department of Computer Engineering, Sharif University of Technology, and IPM School of Computer Science, ramtin@mehr.sharif.edu

†Department of Computer Engineering, Sharif University of Technology, and IPM School of Computer Science, ghodsi@.sharif.edu

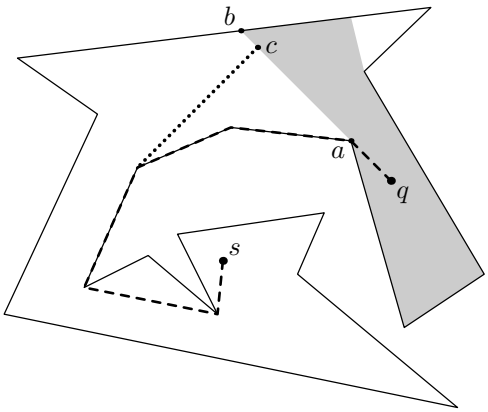

Figure 1: The shaded part is V_q. The window ab separates s and V_q. The shortest path between s and q is shown with heavy dashed segments. c is the point with shortest distance to s that is visible from q.

Outward convexity of the paths from the cusp of the funnel to its endpoints leads to the following observation:

Observation 1 *The sequence of angles between the extension edges and the window w $(\theta_0, \theta_1, \ldots, \theta_k)$ is an increasing sequence.*

We can characterize the optimal contact point $c(w)$ in the following way. For the sequence of angles mentioned in the above observation, one of the following cases holds:

1. There exists an angle $\theta_i = 90°$. In this case, $c(w) = x_i$.
2. There exists a pair of adjacent angles $\theta_i < 90°$ and $\theta_{i+1} > 90°$. In this case, $c(w)$ is the foot of the perpendicular from v_{i+1} to w.
3. All angles in the sequence are greater than $90°$. In this case, $c(w) = a$.
4. All angles in the sequence are less than $90°$. In this case, $c(w) = b$.

3 The Algorithm

When receiving a query point q, we take the following two steps:

1. Compute the window w that separates s from V_q.
2. Compute the optimal point $c(w)$.

Both steps must be done in logarithmic time to have an efficient query-time. To find the window in the first step, observe that the window separating s from V_q is specified by the last vertex of P of the shortest path from s to q (Fig. 1). Thus, having computed

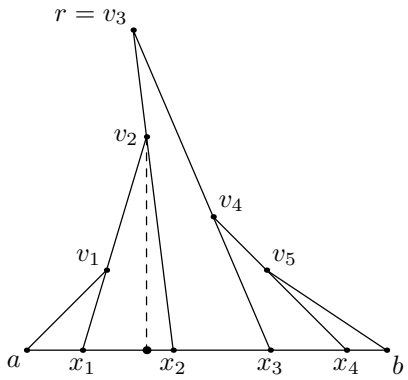

Figure 2: The funnel $F(w)$ over the window $w = ab$. The optimal point $c(w)$ is the foot of the perpendicular from v_2 to ab.

SPM(s) during the preprocessing phase, we can find the window w in $O(\log n)$ time using a standard point-location algorithm. For the second step, we must precompute the optimal contact point on all segments in P that can be a window of a query point. 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 from V_x.

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 which 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. 3(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 vertex the shortest path $\pi(s, a)$ passes through. 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 the extension of e_1 and e_2 where e_1 and e_2 are the edges incident to a (Fig. 3(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 the x. We denote this set of separating windows by $Sep(a)$.

For a reflex vertex a, our goal is to partition $Sep(a)$ into a number of sets such that knowing the set w belongs to, we can find the optimal point $c(w)$ efficiently. To do this, we use a radial sweep around a. There are two kinds of events in the sweep process: angles at which the last vertex of $\pi(s, c)$ changes (*interval events*), and angles at which the structure of the funnel changes (*funnel events*). These events defines the desired partition. We consider the two types of events subsequently.

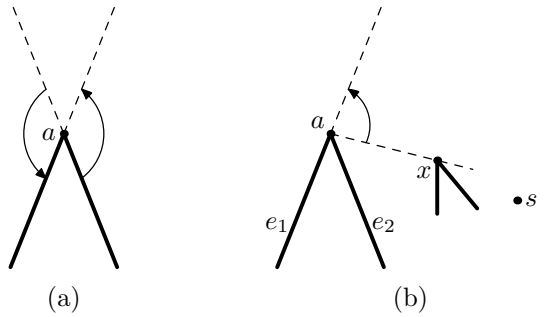

Figure 3: (a) Two angular intervals defining windows around a reflex vertex a. (b) The single interval defining separating windows around a. x is the last vertex on $\pi(s, a)$.

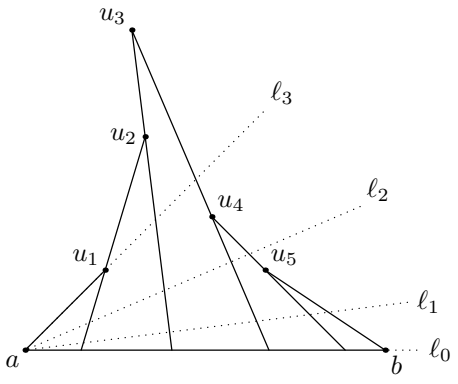

Figure 4: Partition of the separating windows around a. The angular interval between ℓ_0 and ℓ_3 defines the set of all separating windows. ℓ_1 and ℓ_2 are perpendiculars to v_3v_2 and v_3v_4 extensions respectively.

3.1 Interval Events

Suppose $Sep(a)$ is defined by the angular interval $[\alpha, \alpha']$. For $\alpha \leq \varphi \leq \alpha'$, let w_φ denote the window with endpoint a and angel φ. Consider the perpendiculars from a to two adjacent extension edges v_ix_i and $v_{i+1}x_{i+1}$. For a window w_ϕ that lies between these two perpendiculars, we can say $\theta_i < 90°$ and $\theta_{i+1} > 90°$. So, the optimal point $c(w_\phi)$ is the foot of the perpendicular from v_{i+1} to w_ϕ. For example, in Figure 4, ℓ_1 and ℓ_2 are perpendiculars to the extensions of v_3v_2 and v_3v_4 respectively. Hence, for an arbitrary window between ℓ_1 and ℓ_2, the optimal point is the foot of the perpendicular from v_3 to the window. This way, the set of perpendiculars to the extension edges defines the interval events in which the last vertex of $\pi(s, c)$ changes. Since rotating a window towards the cusp of the funnel reduces the angles made between the extension edges and the window, the number of interval events is bounded by $O(n)$.

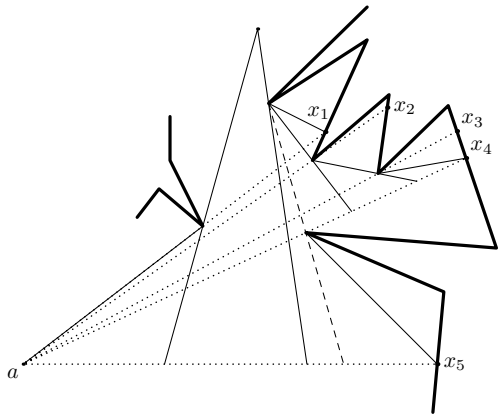

Figure 5: Funnel events for a reflex vertex a. The angular interval between ax_1 and ax_5 defines the set of all separating windows. The sweep starts from ax_1. Thick solid lines are the edges of P, thin solid lines are the extension segments, dotted lines show the swap window in different positions, and the dashed line is an extension segment added after visiting ax_4.

3.2 Funnel Events

Observe that the set of separating windows ($Sep(a)$) 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 SPT(s) is computed priory as well as 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 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 was not a leaf before adding the new vertex. For example, in Figure 5, 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. 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* [2], the list of vertices in the funnel can be arranged in sorted order in linear time.

For a vertex a, we have a set of $O(n)$ angular intervals in sorted order so that upon receiving a query window, we can find the interval it belongs to using binary search. Finding the interval the window belongs to, we can compute its optimal point in constant time.

Summarizing the above discussions, we take the following steps during the preprecessing phase:

1. Compute SPT(s) and SPM(s)
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.

The first step can be done in $O(n)$ time using the algorithm of [1]. 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 [7, 3]. 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 $O(n)$ in total and needs constant time per event. This leads to $O(n)$ preprocess for each reflex vertex. The partition for the reflex vertex is of size $O(n)$. As there are $O(n)$ reflex vertices, the total preprocessing time is $O(n^2)$ and $O(n^2)$ space is needed to store the partitions.

Upon receiving a query, we find the window ($w = ab$) separating s from the query point q in $O(\log n)$ time (by finding q in SPM(s)). We perform a binary search on the partition associated with the reflex vertex a. Finding the optimal point $c(w)$ takes constant time. So we have our main result as the following:

Theorem 1 *Given a simple polygon P and a source point s inside P, we can preprocess the input in $O(n^2)$ time and the same space to answer the queries of this type in $O(\log n)$ time: given a query point q invisible from s, find the point c with minimum distance to s that is visible from q. The path from s to c can be reported in additional time proportional to the length of the path.*

4 Conclusion

We presented an algorithm to preprocess the input polygon in $O(n^2)$ time and space to answer the queries to find the shortest distance to a point visible from a query point in logarithmic time. Possible extensions to this problem involves studying the problem when the path is required to end to a given target point (t) and the query point must be seen from a point during the path. This makes us study the behavior of the optimal point based on two the funnels related to s and t made over the window which is not a direct extension of the behavior regarding one funnel. Another extension 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.

References

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

[2] L. J. Guibas, E. McCreight, M. Plass, and J. Roberts. A new representation for linear lists. In *Proc. 9th Annu. ACM Sympos. Theory Comput.*, pages 49–60, 1977.

[3] B. Joe and R. B. Simpson. Correction to Lee's visibility polygon algorithm. *BIT*, 27:458–473, 1987.

[4] R. Khosravi and M. Ghodsi. Shortest paths in simple polygons with polygon-meet constraints. *Inform. Process. Lett.*, 91:171–176, 2004.

[5] R. Khosravi and M. Ghodsi. Shortest paths with single-point visibility constraints. *submitted to Scientia Iranica*, 2004.

[6] R. Khosravi, M. Ghodsi, and M. Taghdiri. Shortest point-visible paths on polyhedral surfaces. In *Proc. of the 10th International Conference on Computing and Information (ICCI'2000)*, 2000.

[7] D. T. Lee. Visibility of a simple polygon. *Comput. Vision Graph. Image Process.*, 22:207–221, 1983.