FREIE UNIVERSITÄT BERLIN

Shortest Inspection-Path

Queries in Simple Polygons

Christian Knauer, Günter Rote

B 05-05
April 2005

FACHBEREICH MATHEMATIK UND INFORMATIK
SERIE B · INFORMATIK

Shortest Inspection-Path Queries in Simple Polygons

Christian Knauer, Günter Rote

Institut für Informatik, Freie Universität Berlin,
Takustraße 9, D–14195 Berlin, Germany.
{knauer,rote}@inf.fu-berlin.de

Abstract. We want to preprocess a simple n-vertex polygon P to quickly determine the shortest path from a fixed source point $s \in P$ to *some* point visible from a query point $q \in P$. We call such queries *inspection-path queries*. We give an algorithm that computes a data structure which answers the queries in logarithmic time. The data structure has $O(n)$ size and can be computed in $O(n \log n)$ time.

Keywords: Computational geometry, Simple polygons, Shortest paths, Visibility.

1 Introduction

Many variations of the problem of computing shortest paths in *simple* polygons have been studied extensively in the past. One incarnation of the problem is to find the shortest path from a given fixed source point s in a simple polygon P with n vertices to view a query point $q \in P$.

Our goal is to preprocess the input (P, s) to answer queries of this type: Given a query point $q \in P$, find the shortest distance one needs to travel in P from s to see q, i.e., we want to find a point $c \in P$ visible from q that has the shortest distance from s. The query can be answered in $O(n)$ time without preprocessing [4], and in $O(\log n)$ time with $O(n^2)$ preprocessing time and space [5]. We improve and simplify the latter result and describe a solution with $O(n \log n)$ preprocessing time and $O(n)$ space that achieves $O(\log n)$ query time. Our approach can report the shortest path π from s to c in $O(\log n + k)$ time, where k is the length of π. The results are summarized in the following

Theorem 1. *Let P be a simple polygon with n vertices and $s \in P$. We can compute a data structure of $O(n)$ size in $O(n \log n)$ time to answer queries of the following type in $O(\log n)$ time: Given a query point $q \in P$, find the shortest distance one needs to travel in P from s to see q. If desired, the actual shortest path can then be reported in time linear in its length.*

The paper is structured as follows: In Section 2 we define the necessary terminology and give a structural characterization of the solution that forms the basis of our approach. We then prove Theorem 1 by describing and analyzing the data structure and the query process in Section 3. We provide some conclusions in Section 4.

2 Preliminaries

Our terminology and notation follows [5]. The visibility polygon of a point $x \in P$ will be denoted by $V(x)$, the shortest path in P between two points $x, y \in P$ by $p(x, y)$, the shortest path tree from s in P by T, and the shortest path map of P with respect to s by M.

If we remove $V(q)$ from P, the polygon splits into disconnected regions that we call invisible regions. Each such region has exactly one edge in common with $V(q)$ and will be called a *window*. If q is invisible from s, then s lies in an invisible region (if q is visible from s, then clearly $c = q$). In this case it is easy to see that the point c lies on the window w separating s from $V(q)$, in particular c is the point on w that has the shortest distance to s, cf., Fig. 1.

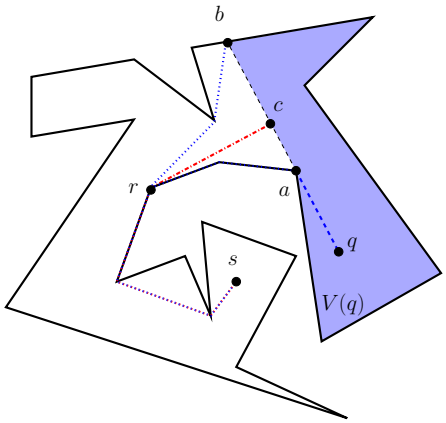

Fig. 1. The window ab separates s and $V(q)$. c is the point with shortest distance to s that is visible from q. The drawing shows the shortest paths from q to a, from q to b, and from q to w, respectively.

We next describe a simple characterization of c given in [5]. To this end let a and b denote be the endpoints of w, and r be the lowest common ancestor of a and b in T, i.e., the last common vertex between the two paths $p(s, a)$ and $p(s, b)$, cf., Fig. 1.

The paths $p(r, a), p(r, b)$ together with the segment $w = ab$ form the *funnel* of w which will be denoted by F, cf. Fig. 2. Note that the paths $p(r, a), p(r, b)$ are outward convex.

Let $a = v_0, v_1, \ldots, r = v_m, \ldots, v_k, v_{k+1} = b$ denote the vertices of the funnel from a to b. F can be decomposed into triangles by extending the edges of F until they intersect w. Let x_i denote the intersection point of the extension of the edge $v_i v_{i+1}$ with w (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 of P. Denote

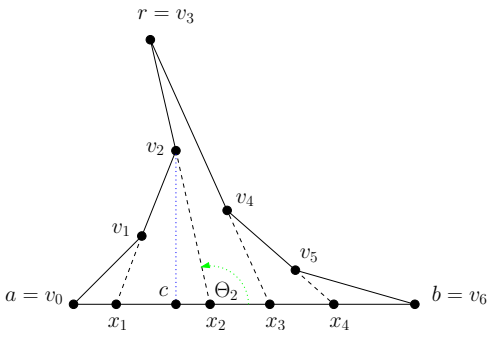

Fig. 2. The funnel F over the window $w = ab$. The optimal point c is the foot of the perpendicular from v_2 to w.

the angles between the extension edges and the window by $\theta_0, \theta_1, \ldots, \theta_k$, i.e., $\theta_i = \angle b x_i v_i$ for $0 \leq i < k$ and $\theta_k = \pi - \angle a b v_k$.

The outward convexity of the paths $p(r, a), p(r, b)$ implies that the sequence $\theta_0, \theta_1, \ldots, \theta_k$ is increasing. As a consequence we can characterize the optimal contact point c in the following way:

1. $\theta_i = \pi/2$ for some $0 \leq i \leq k$. In this case, $c = x_i$.
2. $\theta_i < \pi/2$ and $\theta_i + 1 > \pi/2$ for some $0 \leq i \leq k$. In this case, c is the foot of the perpendicular from v_{i+1} to w.
3. $\theta_0 > \pi/2$. In this case, $c = a$.
4. $\theta_k < \pi/2$. In this case, $c = b$.

We can therefore search for c by looking at the angles θ_i: If $\theta_i > \pi/2$ then c lies left of x_i, whereas if $\theta_i < \pi/2$ then c lies right of x_i.

3 The data structure

To answer a query q we will proceed in two steps:

1. Compute the window w that separates s from $V(q)$ along with the funnel base r.
2. Compute the optimal point c on w.

After the preprocessing phase, the first step can be done in $O(\log n)$ time using standard data structures for point-location and ray-shooting queries, and for lowest common ancestor (LCA) queries in trees. The second step involves a binary search and the total running time for this step will also be $O(\log n)$.

3.1 Preprocessing phase

The preprocessing phase prepares two data structures: the first set of data structures will enable us to quickly compute the window w that separates s from $V(q)$ along with the funnel base r in the query phase. Another data structure will help us with the binary search for the optimal point c on w.

Computing the funnel. In the first step of the preprocessing phase we do the following:

1. Compute the shortest path tree T and the shortest path map M in $O(n)$ time, along with a point-location data structure for M that supports $O(\log n)$ time point-location queries [1, 6].
2. Compute in $O(n \log n)$ time a data structure that supports $O(\log n)$ time ray-shooting queries to P [3].
3. Preprocess T in $O(n)$ time to support $O(1)$ time LCA-queries [2].

Computing the optimal point on the window. In the second step of the preprocessing phase we do the following:

1. In $O(n \log n)$ time store the $O(n)$ vertices V_M of the shortest path map M in an array A, sorted in clockwise order along the boundary of P. Note that these vertices are defined by looking at the extensions of the edges of P until they hit the boundary of P, see Fig. 3.
2. Compute a tree T' in $O(n)$ time by augmenting the tree T as follows:
   (a) Add all the vertices of V_M as new leaves.
   (b) Add an edge between a vertex u of T and a vertex $v \in V_M$ iff u is the endpoint of an edge e of P and the extension of e beyond u that hits the boundary in v.
3. Preprocess T' in $O(n)$ time to support $O(1)$ time LCA-queries.

3.2 Query phase

As a first step in the query phase we will check if q is visible from s (in this case $c = q$). This can be done in $O(\log n)$ time by shooting a ray from q in the direction of s and testing if the boundary of P is hit before s. In the following we can assume, that q is not visible from s.

As we already mentioned, the query will then be answered in two steps: First we compute the window w that separates s from $V(q)$ along with the funnel base r, then we find the optimal point c on w.

Computing the funnel. To find $w = ab$ and r in $O(\log n)$ time in the first step of the query phase we proceed as follows:

Since the window separating s from $V(q)$ is specified by the last vertex of P on the shortest path from s to q (Fig. 1), we can find a in $O(\log n)$ time by locating q in M. To find b in $O(\log n)$ time we shoot a ray from q in the direction of a. Next, we compute v_k (which is an endpoint of the edge containing b) in $O(1)$ time, and finally, we get the funnel base $r = LCA_T(a, v_k)$ in $O(1)$ time.

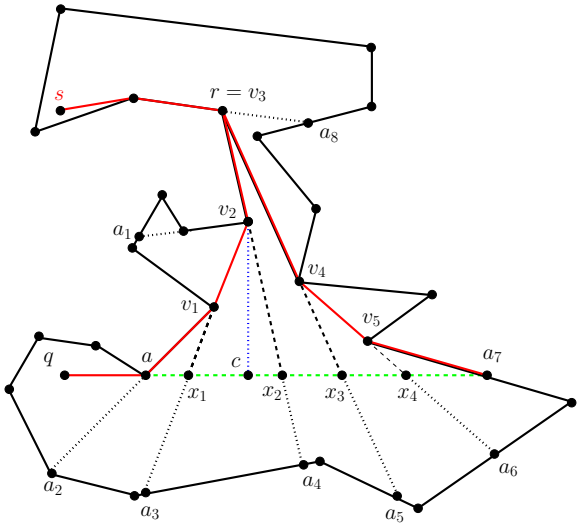

Fig. 3. The vertices $a_1, \ldots, a_8$ are created by the extensions of edges of P that hit the boundary. They are the vertices of the shortest path map of P that are not polygon vertices.

Computing the optimal point on the window. To find the optimal point c on w in $O(\log n)$ time in the second step of the query phase we proceed as follows:

- First we check if $\theta_0 > \pi/2$ or $\theta_k < \pi/2$. In the first case $c = a$, in the second case $c = b$, and in either case we are finished.
- Next we look at the extensions of the edges emanating from the apex $r = v_m$ of the funnel. If $\theta_{m-1} = \pi/2$, or $\theta_m = \pi/2$, or $\theta_{m-1} \leq \pi/2 < \theta_m$, c is the foot of the perpendicular from v_m to w and we are finished.
- If $\theta_{m-1} > \pi/2$, then $\theta_i > \pi/2$ for $m \leq i \leq k$, since the angle sequence is increasing. In particular c is the foot of the perpendicular from some vertex v_i to w, where v_i is on the left side $p(r, a)$ of the funnel F, i.e., $1 \leq i < m$. To determine for which vertex v_i the perpendicular to w has to be drawn, we would like to perform a binary search on the sequence $v_0, \ldots, v_k$. However this sequence is not directly accessible, so we use the array A instead, and perform a binary search on the interval between a and b in this array. For a vertex u in this interval we compute $LCA_{T'}(v_1, u)$, which is one of the vertices $v_1, \ldots, v_m$ on the left edge of the funnel, say v_i. By computing the angle θ_i we can decide if the binary search has to continue to the left or to the right of u. After $O(\log n)$ iterations the binary search is narrowed down to an interval between two successive vertices in A. This implies that the point v_i from which the perpendicular to c has to be drawn is also determined.
- In the case that $\theta_{m-1} < \pi/2$, we have that $\theta_i < \pi/2$ for $1 \leq i < m$, so c is the foot of the perpendicular from some vertex v_i to w, where v_i is on the right side $p(r, b)$ of the funnel F. In this case, we proceed analogously to

find v_i by binary search in A. Again, the total running time for the search is $O(\log n)$.

In the end, we can compute the length of the shortest path in constant time from the information stored in the shortest path tree. If desired, the shortest path itself can be output in linear time.

4 Conclusion

We described a data structure to find for a given query point q in a simple n-vertex polygon P with a designated starting point s the shortest path from s to some point visible from q. Our approach yields a structure of linear size that can be computed in $O(n \log n)$ time and achieves logarithmic query time. This significantly improves previous work on the subject [5]. Our approach also seems to be conceptually simpler. It remains unclear if the problem can still be solved efficiently if we consider the case of polygonal scenes, i.e., polygons with holes.

Acknowledgements

We would like to thank Frank Hoffmann for fruitful discussions on the subject.

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. D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.
3. J. Hershberger and S. Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *J. Algorithms*, 18:403–431, 1995.
4. R. Khosravi and M. Ghodsi. Shortest paths in simple polygons with polygon-meet constraints. *Inf. Process. Lett.*, 91(4):171–176, 2004.
5. R. Khosravi and M. Ghodsi. The fastest way to view a query point in simple polygons. In *Abstracts of the 21st European Workshop on Computational Geometry (EWCG), Eindhoven, Netherlands*, pages 187–190, 2005.
6. D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12(1):28–35, 1983.