Shortest Paths with Single-Point Visibility Constraint

Ramtin Khosravi* Mohammad Ghodsi†

Department of Computer Engineering
Sharif University of Technology

Abstract

This paper studies the problem of finding a shortest path between two points in presence of single-point visibility constraints. In this type of constraints, there should be at least one point on the output path from which a fixed viewpoint is visible. The problem is studied in various domains including simple polygons, polygonal domains, and polyhedral surfaces. The method is based on partitioning the boundary of the visibility region of the viewpoint into a number of intervals. This is done from the combinatorial structure of shortest paths from source and destination to the points on the boundary. Our result for the case of simple polygons is optimal with $O(n)$ time bound. The running time for the cases of polygonal domains, convex and non-convex polyhedral surfaces are $O(n^2)$, $O(n^2)$, and $O(n^3)$ respectively.

Keywords: computational geometry, shortest path, visibility.

1 Introduction

Finding a shortest path between two points is a basic problem in computational geometry and has many applications in different areas such as motion planning and navigation. The problem has been studied over various geometric domains such as simple polygons [1], polygonal domains [2, 3], and polyhedral surfaces [4, 5, 6]. Also, several variations exist depending on the metric used for computing distances, and different constraints applied to the solution path. Examples of such restrictions are curvature constraints [7], or altitude constraints [8]. The visibility constraints are less studied so far. In this type of constraints, the path is required to satisfy some visibility properties, e.g. the entire or parts of the path should be visible from a given viewpoint. Applications for this constraint are mainly in communication systems where direct visibility is needed, or in guarding problems. An example of a motion planning problem combined with visibility constraints can be found in [9] in which the problem of locating a continuously-moving target using a group of guards moving inside a simple polygon is studied.

Single-point visibility constraint requires the path to include at least one point from which a given viewpoint is visible. In this paper, we study the problem of finding a shortest path with single-point visibility constraint in several domains, including simple polygons, polygonal domains, and polyhedral surfaces. The algorithms we propose for the cases of simple polygons and polygonal domains run in $O(n)$ and $O(n^2)$ time bounds respectively. The authors have studied the case of polyhedral surfaces in [10] and proposed an algorithm with $O(n^2 \log n)$ and $O(n^3 \log n)$ time bounds for convex and non-convex cases respectively. The extra $O(n)$ factor in the latter case comes from the fact that the visibility region for a viewpoint on a non-convex surface has $O(n)$ components of $O(n)$ edges. In this paper, we improve the time bound for both cases. We do this for the convex case by a more accurate analysis of the previous algorithm to obtain a running time of $O(n^2)$. Also, an improvement to the algorithm for the non-convex case yields a $O(n^3)$ time bound.

For the case of simple polygons, the authors have studied a more general version of the problem in which the goal is to find the shortest path in presence of *polygon meet* constraints [11]. In this type of problem, the path is constrained to have non-empty intersection with a *target* polygon inside the input simple polygon. The problem is solved in $O(n)$ time where n is the number of vertices in

*`ramtin@mehr.sharif.edu`

†`ghodsi@sharif.edu`

both target and input polygons. Visibility constraints can be viewed as a special case of polygon meet constraints if we consider the visibility polygon of the viewpoint as the target polygon. So, the problem studied in this paper for the case of simple polygons is solvable by the method of [11], but the case is included in this paper too to show how the general approach presented to solve the other two cases can be used to solve the case of simple polygons. Note that this approach solves the case for simple polygons in a more simple way by considering special properties of visibility polygons.

If we have to visit multiple viewpoints during motion along the path, we face a related problem called *TSP with Neighborhoods* in which multiple polygonal regions, called neighborhoods, are given and the goal is to find a tour that visits every neighborhood. The problem is NP-hard [12], and several approximation algorithms have been presented for different cases [13, 14, 15]. Recently, Dror et. al [16] have presented an algorithm for the problem of finding a shortest path that visits k given convex polygons in a given order. Also, they have shown that the problem is NP-hard for the case of non-convex polygons.

The approaches used in this paper for different domains have similar structure, so we formulate them in a generic form in section 2, then discuss issues specific to the cases of simple polygons, polygonal domains, and polyhedral surfaces in sections 3, 4, and 5 respectively.

2 The General Approach

In this section, we consider the general approach for finding a shortest path between two points with the constraint that at least one point on the path is visible from a given viewpoint. Let $\mathcal{P}$ be the geometric domain of the problem under consideration. We consider $\mathcal{P}$ as a set of points, $\mathcal{V}_p$ as the *visibility region* of the given viewpoint $p \in \mathcal{P}$, and $\mathcal{B}_p$ as the boundary of $\mathcal{V}_p$. In all domains considered in this paper, $\mathcal{B}_p$ consists of a number of line segments. The set $\mathcal{P} - \mathcal{V}_p$ consists of a number of connected sets of points, called *invisible regions* of the domain.

We call a path between two points s and t in $\mathcal{P}$ a *p-visible* path if it has non-empty intersection with $\mathcal{V}_p$. Our goal is to compute a shortest path p-visible path between s and t. Note that in polygonal domains and polyhedral surfaces, there may be no "unique" shortest path between two points, so there may be several shortest p-visible paths between s and t in those cases. In our algorithms, we concentrate on finding one of these paths. Let q be the first point visible from p, when walking along a shortest p-visible path from s to t. Obviously, q lies somewhere on $\mathcal{B}_p$, and the subpaths from s to q and from q to t are locally optimal. So, our problem reduces to finding a point q with this property.

The main idea of the algorithm is to partition $\mathcal{B}_p$ into a set of intervals such that for each interval I, we can easily find the point $q(I) \in I$ whose total shortest distance to s and t is minimum among all points on I. Then q is the one with the minimum total shortest distance.

To do this, we use the notion of *interval of optimality* previously used in works on this topic [4]. A connected set of points I on $\mathcal{B}_p$ is an interval of optimality with respect to a point $x \in \mathcal{P}$, if the shortest paths from x to any point inside I have a fixed combinatorial structure. We denote the set of all such intervals by L_x. When it is clear from the context, we may use "interval" instead of the term "interval of optimality". To partition $\mathcal{B}_p$, we compute L_s and L_t and merge the endpoints of the intervals in the two sets to obtain a set of intervals denoted by $L_{s,t}$ (Fig. 1).

Intervals in L_s (resp. L_t) can be found by intersecting $\mathcal{B}_p$ with the edges of the *shortest path map* of s (resp. t), although computing the entire set of intervals may not be necessary. For the case of polyhedral surfaces, we use a subdivision of the surface giving the same information as the shortest path map for the two dimensional cases.

Based on the above definitions, we sketch the generic algorithm for computing a shortest p-visible path between s and t as the following:

1. Compute $\mathcal{B}_p$ (or the relevant portion of it).

2. Compute the set $L_{s,t}$ on the relevant subset of $\mathcal{B}_p$.

3. For each interval $I \in L_{s,t}$, find the point $q(I)$ which has the minimum total distance from s and t.

4. Let q be the point with the minimum total distance from s and t among $\{q(I) : I \in L_{s,t}\}$.

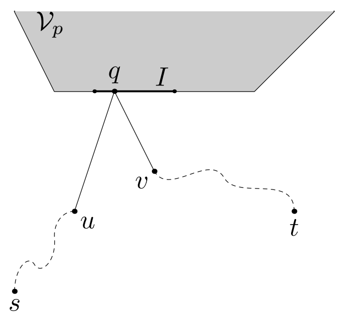

Figure 1: In two dimensional domains, an interval of optimality $I \in L_{s,t}$ (shown with heavy solid line) is a connected subset of $\mathcal{B}_p$ such that shortest paths from s (resp. t) to its points passes through u (resp. v) as the last vertex. $q(I) \in I$ is the point with minimum total shortest path distance to s and t.

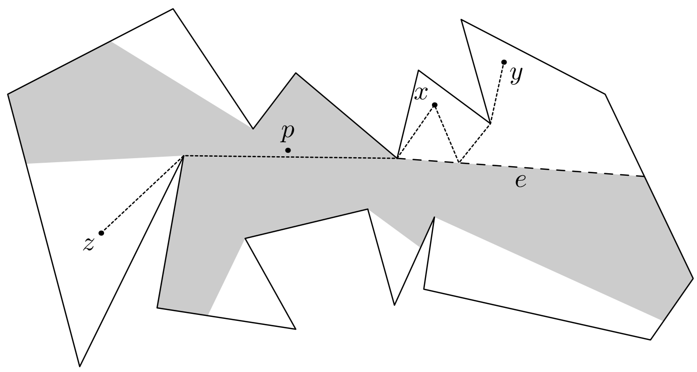

Figure 2: A simple polygon with a viewpoint p inside it. The shaded area is $\mathcal{V}_p$ and there are six invisible regions in the polygon. The shortest path between x and z is already p-visible while the shortest p-visible path between x and y touches e and returns.

5. Report a shortest path from s to q appended by a shortest path from q to t.

The first two steps in the above algorithm depend on the specific geometric domain of the problem, which we consider in the succeeding sections.

3 Simple Polygons

For the case of simple polygons, many shortest path problems have linear algorithms due to the fact that there is exactly one "taut-string" between any two points in a simple polygon that can be found using dual of a triangulation for the polygon, which is a tree. In the method presented by Guibas et al. [1], one can construct the shortest path map of a given source point using a DFS traversal of the mentioned tree. The method maintains a funnel-like structure during this traversal to construct the shortest path map. We will consider how to use this structure to find the intervals of optimality on the relevant portion of $\mathcal{B}_p$ in linear time.

In this case, $\mathcal{P}$ is supposed to be a simple polygon with n edges. $\mathcal{B}_p$ is also a simple polygon whose edges are extensions of segments connecting p to the vertices of $\mathcal{P}$ visible from p. Hence, each edge of $\mathcal{B}_p$ decomposes $\mathcal{P}$ into two parts, one contains $\mathcal{V}_p$ and the other is an invisible region (Fig. 2). If either s or t lies inside or on the boundary of $\mathcal{V}_p$, the shortest path between s and t is already p-visible. The same is true if s and t lie in two different invisible regions (like the points x and z in Fig. 2). This is due to the fact that the only way for the path to exit from an invisible region is to cross an edge of $\mathcal{V}_p$. So, we can assume that the points s and t are both in one invisible region. We name this invisible region W. Note that testing the above conditions can be done in $O(n)$ total time.

Based on the above assumption, we can restrict the computations to the relevant portion of $\mathcal{B}_p$

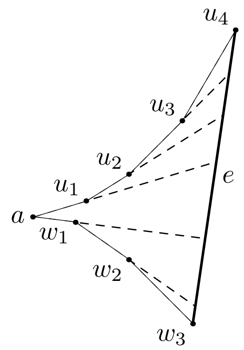

Figure 3: Intervals of optimality on the edge e obtained by running a shortest path algorithm in the polygon W.

which is a single edge e common to $\mathcal{V}_p$ and W. It can be easily verified that in this case, the shortest p-visible path does not enter $\mathcal{V}_p$ and just *touches* $\mathcal{V}_p$ and returns (like the path between x and y in Fig. 2). The reason is obvious, since we can take a shortcut along e between the point that the path enters $\mathcal{V}_p$ and the point that it exits from. So, our problem is to find the set of intervals of optimality on the edge e.

Since the shortest path does not exit W, we can find the intervals on e easily using the shortest path algorithm of [1] to construct the shortest path map of a simple polygon. We consider W as the input polygon to the mentioned algorithm with s as the source point. During the DFS traversal, we finally visit the boundary edge e (Fig. 3). Assuming the funnel-like structure maintained during the traversal has the form $[u_l, u_{l-1}, \ldots, u_1, a, w_1, \ldots, w_k]$ at that time with $a = u_0 = w_0$ as its cusp (thus $e = u_l w_k$), the rays emanating from u_i (resp. from w_i) and passing through u_{i+1} (resp. through w_{i+1}) partition e into $k+l-1$ intervals. Each interval has the property that the last polygon vertex on the shortest paths from s to all of its points is the same. These intervals form the set L_s.

Computing the set $L_{s,t}$ for this case is easy now. We have to compute L_s and L_t using the method mentioned above, and a merging process is required to obtain $L_{s,t}$ from the computed intervals. Assuming the shortest Euclidean distances from s and t to all vertices of $\mathcal{P}$ have already been computed, we have the following lemma:

Lemma 3.1 *Let I be an interval in $L_{s,t}$. One can find the point $q(I)$ on I with minimum total distance to s and t in constant time.*

Proof. I has the property that the last vertices of the shortest paths from s and t to an arbitrary point on I are the same for all points of I. Let u (resp. v) be the last vertex on shortest paths form s (resp. t) to the points of I. To find $q(I)$, we can reflect v about the line supporting I and connect the reflected point to u. If the segment obtained in this way intersects I, the intersection point will be $q(I)$. Otherwise, $q(I)$ will be one of the endpoints of I depending on which side of I lies the intersection point between the segment and the line supporting I. □

Based on the above discussions, the following steps are taken to compute the shortest p-visible path in a simple polygon:

1. If p is visible from either s or t, report the shortest path between s and t.
2. Compute the visibility polygon of p ($\mathcal{V}_p$).
3. Compute the invisible regions in which s and t lie.
4. If the invisible regions of s and t are different, report the shortest path between s and t, otherwise, name the invisible region in which both s and t lie, as W.
5. Run the shortest path algorithm in the simple polygon W twice, assuming s and t as the source point each time. As the result of this step, the following information is generated:
    - Shortest distances from both s and t to every vertex of $\mathcal{P}$, and

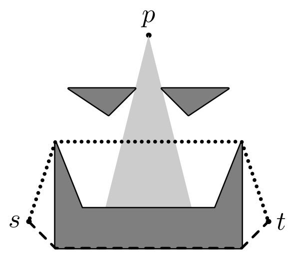

Figure 4: The shortest p-visible path between s and t (shown with dots) crosses part of $\mathcal{V}_p$ (light-shaded) while s and t are in one invisible region and the shortest path between them (dashed line) does not cross $\mathcal{V}_p$. Dark shaded polygons are obstacles.

- The two sets L_s and L_t.

6. Compute $L_{s,t}$ by merging the endpoints of the intervals in L_s and L_t.

7. For each interval $I \in L_{s,t}$ find the point $q(I)$ which has the minimum total distance from s and t. Let q be the point among all $q(I)$ that has the minimum total distance.

8. Report the shortest path from s to q appended by the shortest path from q to t.

To analyze the running time of the algorithm, observe that the check in step 1 of the above algorithm can be easily done in linear time. The visibility polygon computation in step 2 can be done using the linear time algorithm of Lee [17, 18]. To compute the invisible region in which s (resp. t) lies (step 3), we can traverse the boundary of the invisible region, starting from the intersection of the ps (resp. pt) directed half-line and the boundary of $\mathcal{P}$. This can be done in linear time assuming that we have the vertices of $\mathcal{P}$ in order. Step 5 is calling the shortest path algorithm of Guibas et al. [1] twice and can be done in $O(n)$ time. Since the funnel structure is stored in a finger search tree [19] which is based on the B-tree data structure, we can obtain the sorted list of intervals in L_s and L_t both in $O(n)$ time. Therefore, $L_{s,t}$ can be computed in linear time (step 6). Finally, the minimum point q can be computed in $O(n)$ time according to lemma 3.1. Each step of the algorithm uses at most $O(n)$ space, hence the overall algorithm needs linear time and space. Thus, we have the following result for the case of simple polygons:

Theorem 3.1 *Given a pair of points s and t and a viewpoint p inside a simple polygon, the shortest p-visible path from s to t inside the polygon can be computed in $O(n)$ time and $O(n)$ space.*

4 Polygonal Domains

In this case, $\mathcal{P}$ is supposed to be a polygonal domain with total number of n edges. The problem is to find a shortest obstacle-avoiding p-visible path between s and t. We assume that the domain is bounded by a given simple polygon with a number of (simple, non-overlapping) polygonal obstacles inside. By *free space* we mean the set of points inside or on the bounding polygon minus the interior of the obstacles. The shortest path map of the free space can be computed using the algorithm of Hershberger and Suri [3] in the worst-case optimal time $O(n \log n)$ using $O(n \log n)$ space, where n is the total number of vertices of the domain. The subdivision can be used to answer single-source shortest path queries using classic point-location algorithms in logarithmic time.

As stated earlier, the main challenge in a particular domain is to find the set of intervals of optimality efficiently. Taking the same approach as the case of simple polygon, we must find the invisible region in which s and t lie (W), and run a shortest path algorithm (such as [3]) to construct the shortest path map of W and take the intervals made on the edges common to W and $\mathcal{V}_p$. However, unlike the case of simple polygons, the boundary between W and $\mathcal{V}_p$ may consist of more than one edge. So, there may be cases that the only shortest p-visible path between s and t enters $\mathcal{V}_p$ from one edge and exit from another, while s and t both lie in one invisible region W and the only shortest path between them lies completely inside W (Fig. 4).

Based on the above observation, we use the general approach presented in Section 2 without major modifications for the case of polygonal domains. To compute L_s and L_t, we compute the

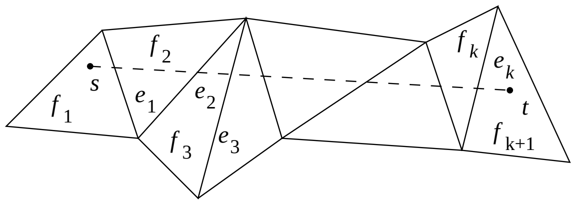

Figure 5: Shortest path from s to t unfolded along its edge sequence

shortest path map of the domain twice with respect to the points s and t, and find the intersections of the two maps with $\mathcal{B}_p$. To find the intersections, we can use known algorithms for subdivision overlay such as the algorithm of Finke and Hinrichs [20] which solves the problem in optimal time $O(n+k)$ where k is the number of intersections which is $O(n^2)$ in the worst case. So, the method computes L_s and L_t in $O(n^2)$ time. Note that to compute $L_{s,t}$, we have to merge the endpoints of the intervals in L_s and L_t which requires having the intervals in each set in sorted order. The algorithm of Finke and Hinrichs produces the output subdivision in a *quad view* data structure [21] which allows ordered traversal of the intervals in L_s or L_t using the operations provided for traversal of vertex rings and edge rings in the output subdivision.

Now we analyze the running time of the algorithm in this case. Computing the visibility polygon can be done using the optimal algorithm of Heffernan and Mitchell [22] which requires $O(n+h\log h)$ time (h is the number of obstacles in the domain which is $O(n)$ in worst case). Constructing the shortest path maps takes $O(n\log n)$ time and the same space. Computing L_s and L_t is done in $O(n^2)$ based on the preceding lemma. Finally, the minimum point q can be computed in $O(n^2)$ time, since the size of $L_{s,t}$ is $O(n^2)$, and the property stated in lemma 3.1 holds in this case too. Thus, we have the following result for the case of polygonal domains:

Theorem 4.1 *Given a pair of points s and t and a viewpoint p inside a polygonal domain, a shortest p-visible path from s to t inside the free space can be computed in $O(n^2)$ time and the same space.*

5 Polyhedral Surfaces

In this section, we consider the geometric domain of polyhedral surfaces, i.e. $\mathcal{P}$ is the surface of a polyhedron. In the case of convex polyhedra, $\mathcal{V}_p$ is a connected region whose boundary consists of $O(n)$ edges of $\mathcal{P}$. In non-convex case, $\mathcal{V}_p$ is a set of possibly disconnected regions of total complexity of $O(n^2)$. As the complexity of the visibility region of a point on a (possibly non-convex) polyhedron is quadratic in size of the polyhedron and the algorithms for finding the visibility map are superquadratic in general [23], we assume the visibility region of the point to be seen is determined through a preprocessing stage and focus on finding a shortest p-visible path.

The problem of finding a shortest path between two points on the surface of a polyhedron is well studied. Especially Chen and Han [5] present a method for building a subdivision of the surface which can be used for finding a shortest path from a fixed source to a given query point efficiently. The subdivision can be built in $O(n^2)$ time. The best known algorithm for finding the shortest paths on polyhedral surfaces is [6] which finds a shortest path in $O(n\log^2 n)$ using the *wavefront propagation* method.

5.1 Shortest Paths on a Polyhedron

We briefly review the related terminology borrowed from [4]. Two faces f and f' are said to be *edge-adjacent* if they share a common edge e. A *sequence of edge-adjacent faces* is a list of one or more faces $\mathcal{F} = (f_1, f_2, \ldots, f_{k+1})$ such that f_i is edge-adjacent to f_{i+1} (sharing a common edge e_i). We refer to the (possibly empty) list of edges $\mathcal{E} = (e_1, e_2, \ldots, e_k)$ as an *edge sequence* and to the vertex of face f_1 opposite e_1 as the *root* of $\mathcal{E}$ (Fig. 5).

Each face has a two dimensional coordinate system associated with it. If faces f and f' are edge-adjacent sharing edge e, we define the *planar unfolding* of face f' onto face f as the image of points of f' when rotated about the line through e into the plane of f such that the points of f' fall on the *opposite* side of e to points of f. Extending this notation, we say that we *unfold* an edge

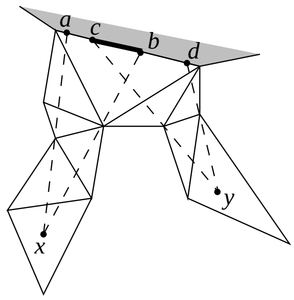

Figure 6: ab is an interval in L_x and cb is an interval in $L_{x,y}$.

sequence $\mathcal{E} = (e_1, e_2, \ldots, e_k)$ as follows: Rotate f_1 around e_1 until its plane coincides with that of f_2, rotate f_1 and f_2 around e_2 until their plane coincides with that of f_3, continue in this way until all faces $(f_1, f_2, \ldots, f_k)$ lie in the plane of f_{k+1}.

We say a path π connects the edge sequence $\mathcal{E} = (e_1, e_2, \ldots, e_k)$ if π consists of segments which join interior points of $e_1, e_2, \ldots, e_k$ (in that order). A path on $\mathcal{P}$ is called geodesic if it is locally optimal and cannot be shortened by small perturbations. The following lemma characterizes such paths:

Lemma 5.1 (Mitchell, Mount and Papadimitriou [4]) *The general form of a geodesic path is a path which goes through an alternating sequence of vertices and (possibly empty) edge sequences such that the unfolded image of the path along any edge sequence is a straight line segment and the angle of the path passing through a vertex is greater than or equal to π. The general form of an optimal path is the same as that of a geodesic path, except that no edge can appear in more than one edge sequence and each edge sequence must be simple.*

5.2 Computing the Shortest p-Visible paths

Consider a maximal set of points on $\mathcal{B}_p$ whose shortest paths to a point x connect the same edge sequence. Such points form an interval that belongs to L_x. The set $L_{x,y}$ is defined similarly (Fig. 6).

To compute the set L_x we use a subdivision presented in [5] which decomposes the surface to a number of regions such that a shortest path from x to a point inside one region has a fixed combinatorial structure (i.e., connects the same edge-sequence). This subdivision plays a role similar to shortest path map in two dimensions. To obtain the subdivision, we cut the surface of $\mathcal{P}$ along the shortest paths from x to all of the vertices of $\mathcal{P}$. The resulting surface can be laid out on a common plane. The layout obtained in this manner is called the *inward layout* of $\mathcal{P}$ (also called *star unfolding* [24]). The vertices of this polygon are the vertices of $\mathcal{P}$ together with the *images of the source point* x under different unfoldings and the edges are shortest paths from the source to the vertices of $\mathcal{P}$. A subdivision of the inward layout can be obtained by constructing the Voronoi diagram on the layout with respect to the images of the source point (Fig. 7). This subdivision has the property that the points in the same region are closer to the corresponding image of the source than to other images, and their shortest paths from the source pass through the same edge sequence. The set L_x is obtained by intersecting $\mathcal{B}_p$ with the edges of the subdivision mentioned above considering x as the source point.

To bound the number of intervals in L_x, observe that the subdivision has two kinds of edges that are to be intersected with $\mathcal{B}_p$: shortest paths to vertices (cuts) and the edges of the Voronoi diagram (ridges). The number of these edges is $O(n)$. In the convex case, the edges of $\mathcal{B}_p$ are edges of $\mathcal{P}$ too, and may have $O(n^2)$ intersections with cuts and ridges in the worst case. In contrast, for the non-convex polyhedra, $\mathcal{B}_p$ has $O(n)$ components, each with $O(n)$ edges, which are not necessarily parts of the edges of $\mathcal{P}$. In this case, each component may have $O(n^2)$ intersections with cuts and ridges, resulting in $O(n^3)$ intersections in general. So, the size of L_x will be $O(n^2)$ in convex and $O(n^3)$ in non-convex case. Note that it can be shown that these bounds are tight (Fig. 8 shows a convex case).

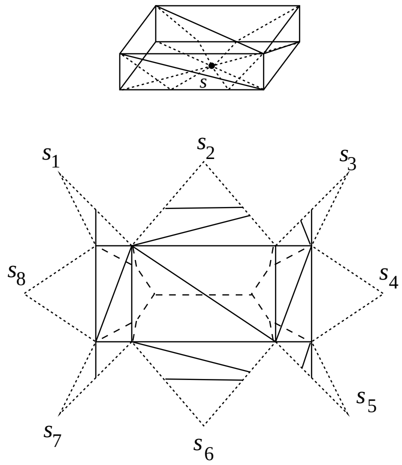

Figure 7: The inward layout of a box. Solid lines are the edges of the subdivision that are part of polyhedron edges. Dashed lines are edges of the Voronoi diagram (ridges) and dotted lines are shortest paths from images of the source to vertices (cuts).

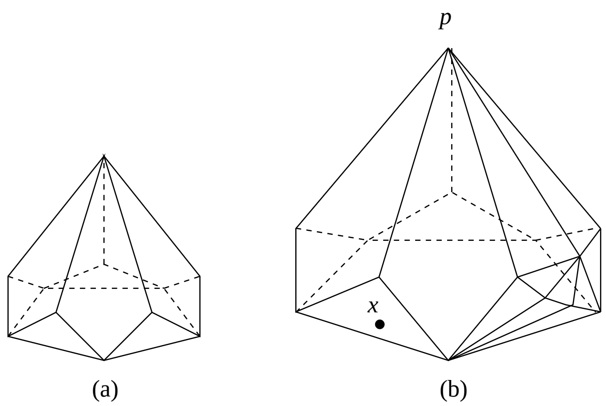

Figure 8: (a) A diamond polyhedron which has $O(n)$ faces (b) Diamond polyhedron with a number of faces added to it.

Table 1: Summary of time and space complexity of the algorithm in various domains

Domain	**Time**	**Space**
Simple Polygons	$O(n)$	$O(n)$
Polygonal Domains	$O(n^2)$	$O(n^2)$
Convex Polyhedral Surfaces	$O(n^2)$	$O(n^2)$
Non-convex Polyhedral Surfaces	$O(n^3)$	$O(n^3)$

For the non-convex case, the inward layout may overlap itself. The algorithm for computing Voronoi diagram, in this case, is slightly different and is given in [5]. In this case, an interval in L_x has the property that there is a vertex v of the polyhedron such that every shortest path from x to a point on the interval passes through v as the last vertex, and the edge sequence from v to points on the interval is the same. As the first part of the path is fixed among the points on interval, given an interval $I \in L_{s,t}$, we can still find the point $q(I)$ whose total distance from s and t is minimum in constant time provided that when computing the intervals, we store the distance to pseudo-source associated with the interval.

Following the general approach stated before, these steps are to be taken to compute a shortest p-visible path between s and t:

1. If p is visible from either s or t, report a shortest Euclidean path between s and t.
2. Run a shortest path algorithm on the surface $\mathcal{P}$ twice, assuming s and t as the source point each time, to find the shortest distance from both s and t to every vertex of $\mathcal{P}$. As a result of this step, two subdivisions of the surface are built with respect to s and t.
3. Compute the set $L_{s,t}$.
4. For each interval $I \in L_{s,t}$ find the point $q(I)$ which has the minimum total distance from s and t. Let q be the point among all $q(I)$ that has the minimum total distance.
5. Report a shortest path from s to q appended by a shortest path from q to t.

To analyze the running time of the algorithm, observe that the check in step 1 of the above algorithm can be done in time proportional to the size of the visibility region which is less than the time bounds stated for the whole algorithm. Computing the subdivisions with respect to s and t takes $O(n^2)$ time for both cases (step 2). Analyzing step 3 (computing $L_{s,t}$) should be done separately for the cases of convex and non-convex polyhedra.

For the convex case, $\mathcal{V}_p$ consists of one or more faces of the polyhedron, so its boundary consists of polyhedron edges. Hence, for an arbitrary x, the size of L_x will be $O(n^2)$. Using the algorithm of Chen and Han, we can find these intervals in $O(n^2)$ time, since the algorithm keeps track of the intersection of shortest paths to vertices with the edges of the polyhedron. Furthermore, the intervals obtained this way are in sorted order since the algorithm keeps the shortest paths to the vertices sorted in angular order around the source point x. Hence, the total time to find L_s and L_t is $O(n^2)$ in this case. Computing $L_{s,t}$ is done by merging the endpoints of the intervals in L_s and L_t which also needs $O(n^2)$ time.

For the non-convex case, $\mathcal{V}_p$ may be disconnected, with total complexity of $O(n^2)$ edges (not necessarily parts of the edges of $\mathcal{P}$). In this case, we cannot use the information about the intersections between the edges of $\mathcal{B}_p$ with those of $\mathcal{P}$ obtained by the shortest path algorithm. To compute L_x, we have to intersect the edges of $\mathcal{B}_p$ with cuts and ridges, yielding to maximum of $O(n^3)$ intervals. This takes $O(n^3)$ time using the same method as the case of polygonal domains. Thus, finding $L_{s,t}$ can be done in $O(n^3)$ time.

Finally, the minimum point q can be computed in $O(n^2)$ (resp. $O(n^3)$) time for the convex (resp. non-convex) case. It can be easily verified that the space complexity of the algorithm is $O(n^2)$ (resp. $O(n^3)$). Thus, we have the following result for the case of polyhedral surfaces:

Theorem 5.1 *Given a pair of points s and t, a viewpoint p, and the visibility region of the viewpoint $\mathcal{V}_p$ on the surface of a polyhedron, a shortest p-visible path from s to t can be computed in $O(n^2)$ time for the convex case and $O(n^3)$ for the non-convex case.*

6 Conclusion

We studied the problem of finding a shortest path between two points with single-point visibility constraint in various domains summarized in Table 1. The time bound for the first case is the same as the bound for the standard shortest path problem (without constraint), so this bounds cannot be improved any further. The case of polygonal domains and polyhedral surfaces may be improved though. The case for polygonal domains can be studied further to see if it is not necessary to construct the entire set of intervals of optimality. Also, it is possible to improve the case for the polyhedral surfaces due to existence of the subquadratic algorithm of Kapoor [6] for finding shortest paths on polyhedral surfaces that uses the wavefront propagation method.

There may be several extensions that can be considered. One extension is to use other metrics for distance computations (e.g. link-distance or weighted region) while having the visibility constraints. For some cases, it is possible to use the same framework as we used in the current paper, i.e. constructing shortest path maps according to the metric used and finding the set of intervals of optimality on the boundary. However, it is possible that for some metric, there may be more specific and more efficient algorithms.

Another extension is to constrain the path to meet an arbitrary general region, not necessarily a visibility region. Generalization of our algorithm to this case is straightforward for some domains (e.g. polygonal domains), since we have not used special properties about visibility regions in those domains. Other domains (e.g. simple polygons) require more study.

Acknowledgment: The authors would like to thank the anonymous referee for his/her valuable comments.

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] J. S. B. Mitchell. Shortest paths among obstacles in the plane. *Internat. J. Comput. Geom. Appl.*, 6:309–332, 1996.

[3] J. Hershberger and S. Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM J. Comput.*, 28(6):2215–2256, 1999.

[4] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou. The discrete geodesic problem. *SIAM J. Comput.*, 16:647–668, 1987.

[5] J. Chen and Y. Han. Shortest paths on a polyhedron. *Internat. J. Comput. Geom. Appl.*, 6:127–144, 1996.

[6] S. Kapoor. Efficient computation of geodesic shortest paths. In *Proc. 32th Annu. ACM Sympos. Theory Comput.*, pages 770–779, 1999.

[7] J. D. Boissonnat, A. Cérézo, and J. Leblond. Shortest paths of bounded curvature in the plane. *Internat. J. Intell. Syst.*, 10:1–16, 1994.

[8] M. De Berg and M. Van Kreveld. Trekking in the alps without freezing or getting tired. *Algorithmica*, 18:306–323, 1997.

[9] A. Efrat, L. J. Guibas, S. Har-Peled, D. C. Lin, J. S. Mitchell, and T. M. Murali. Sweeping simple polygons with a chain of guards. In *Proc. 11th ACM-SIAM Sympos. Discrete Algorithms (SODA'2000)*, pages 927–936, 2000.

[10] 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)*. Unpublished.

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

[12] J. Gudmundsson and C. Levcopoulos. Hardness result for TSP with neighborhoods. Technical Report LU-CS-TR:2000-216, Department of Computer Science, Lund University, Sweden, 2000.

[13] C. Mata and J. S. Mitchell. Approximation algorithms for geometric tour and network design problems. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 360–369, 1995.

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

[15] A. Dumitrescu and J. S. B. Mitchell. Approximation algorithms for TSP with neighborhoods in the plane. In *Symposium on Discrete Algorithms*, pages 38–46, 2001.

[16] M. Dror, A. Efrat, A. Lubiw, and J. S. B. Mitchell. Touring a sequence of polygons. In *Proc. 35th ACM Sympos. Theory Comput.*, 2003.

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

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

[19] 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.

[20] U. Finke and K. Hinrichs. Overlaying simply connected planar subdivisions in linear time. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 119–126, 1995.

[21] U. Finke and K. H. Hinrichs. The quad view data structure: a representation for planar subdivisions. In *Proc. 6th Sympos. Advances in Spatial Databases*, number 951 in Lecture Notes Comput. Sci., pages 29–46, 1995.

[22] P. J. Heffernan and J. S. B. Mitchell. An optimal algorithm for computing visibility in the plane. *SIAM J. Comput.*, 24(1):184–201, 1995.

[23] J. O'Rourke. Visibility. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 25, pages 467–480. CRC Press LLC, 1997.

[24] P. K. Agarwal, B. Aronov, J. O'Rourke, and C. Schevon. Star unfolding of a polytope with applications. Technical Report 031, Dept. Comput. Sci., Smith College, Northampton, MA, July 1993.