Scientia Iranica, Vol. 13, No. 1, pp 25 32

SCIENTIA IRANICA

Shortest Paths with Single-Point Visibility Constraint

R. Khosravi[1] **and M. Ghodsi***

In this paper, the problem of finding the shortest path between two points in the presence of single-point visibility constraints is studied. In these types of constraint, 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 the shortest paths from the source and destination to the points on the boundary. The result for the case of simple polygons is optimal with $O(n)$ time bound. The running time for the cases of polygonal domains and convex and non-convex polyhedral surfaces are $O(n^2)$, $O(n^2)$ and $O(n^3)$, respectively.

INTRODUCTION

Finding the 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 [4-6]. Also, several variations exist, depending on the metric used for computing distances and the different constraints applied to the solution path. Examples of such restrictions are curvature constraints [7] or altitude constraints [8]. The visibility constraints have, so far, been studied less. In this type of constraint, 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, the problem of finding the shortest path with a single-point visibility constraint is studied in several domains, including simple polygons, polygonal domains and polyhedral surfaces. The algorithms proposed 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, the time bound is improved for both cases. This is done 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.

If one has to visit multiple viewpoints during motion along the path, a related problem, called TBP with Neighborhoods is faced, 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 [11] and several approximation algorithms have been presented for different cases [12-

1. *Department of Computer Engineering, Sharif University of Technology, P.O. Box 11365-9517, Tehran, I.R. Iran, and IPM School of Computer Science, Tehran, I.R. Iran.*

*. *Corresponding Author, Department of Computer Engineering, Sharif University of Technology, P.O. Box 11365-9517, Tehran, I.R. Iran, and IPM School of Computer Science, Tehran, I.R. Iran.*

14]. Recently, Dror et al. [15] presented an algorithm for the problem of finding a shortest path that visits k, given convex polygons in a given order. Also, they showed that the problem is NP-hard for the case of non-convex polygons.

The approaches used in this paper for different domains have a similar structure, so, they were formulated in a generic form in the following section. Then, issues specific to the cases of simple polygons, polygonal domains and polyhedral surfaces were discussed, respectively.

GENERAL APPROACH

In this section, a general approach is considered for finding the 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. $\mathcal{P}$ is considered 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, $\mathcal{P} \quad \mathcal{V}_p$, consists of a number of connected sets of points, called invisible regions of the domain.

A path between two points, s and t in $\mathcal{P}$, is called a p-visible path, if it has a non-empty intersection with $\mathcal{V}_p$. The goal of this paper is to compute the 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 the authors algorithms, the concentration is 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, the problem is reduced 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, one 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, the notion of interval of optimality is used, which has been 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. The set of all such intervals is denoted by L_x. When it is clear from the context, one may use "interval" instead of the term of "interval of optimality". To partition $\mathcal{B}_p$, L_s and L_t are computed and the endpoints of the intervals in the two sets are merged to obtain a set of intervals denoted by $L_{s,t}$ (Figure 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, a subdivision of the surface is used giving the same information as the shortest path map for the two-dimensional cases.

Based on the above definitions, the generic algorithm for computing shortest p-visible path between s and t is sketched 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 $\{q(I) : I \in L_{s,t}\}$;
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 is considered in the succeeding sections.

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 the 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

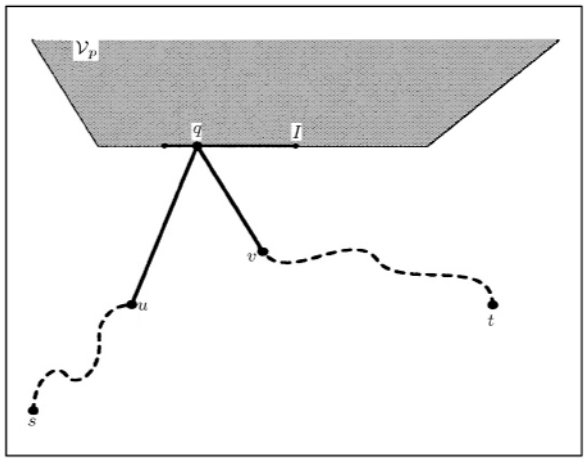

Figure 1. An interval of optimality $I \in L_{s,t}$ (shown with heavy solid line). $q(I) \in I$ is the point with minimum total shortest path distance to s and t.

method maintains a funnel-like structure during this traversal to construct the shortest path map. It will be considered 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 (Figure 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 Figure 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, one can assume that the points s and t are both in one invisible region. This invisible region is named W. Note that testing the above conditions can be done in $O(n)$ total time.

Based on the above assumption, the computations can be restricted to the relevant portion of $\mathcal{B}_p$, 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 $\mathcal{V}_p$, just touches $\mathcal{V}_p$ and returns (like the path between x and y in Figure 2). The reason is obvious, since one can take a shortcut along e between the point where the path enters $\mathcal{V}_p$ and the point where it exits. So, the problem is to find the set of intervals of optimality on the edge e.

Since the shortest path does not exit W, one can find the intervals on e easily using the shortest path algorithm of [1] to construct the shortest path map of a simple polygon. W is considered as the input polygon to the mentioned algorithm, with s as the source point. During the DFS traversal, the boundary edge, e, is finally, visited (Figure 3). Assuming the funnel-like structure maintained during the traversal has the form $[u_l, u_{l-1}, \cdots, u_1, a, w_1, \cdots, w_k]$ at that

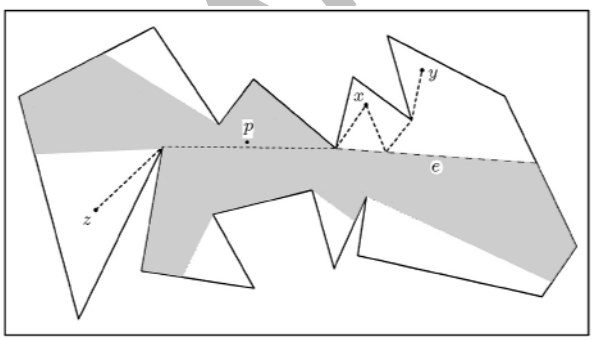

Figure 2. A simple polygon with a viewpoint p inside it. The shaded area is $\mathcal{V}_p$ and there are six invisible regions in the polygon. The shortest path between x and z is already p-visible, while the shortest p-visible path between x and y touches e and returns.

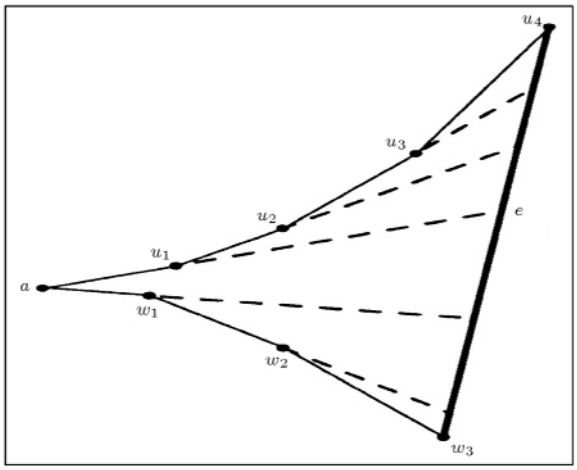

Figure 3. Intervals of optimality on the edge e obtained by running a shortest path algorithm in the polygon W.

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+d}) partition e into $k+l-1$ intervals. Each interval has the property where 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 now easy. One has 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, the following lemma is obtained.

Lemma 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 where 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 the shortest paths to form s (resp. t) to the points of I. To find $q(I)$, v can be reflected about the line supporting I and the reflected point be connected 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 $\mathcal{P}$;
   - The two sets L_s and L_t;
6. Compute $L_{s,t}$ by merging the end points 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. Step 2 can be done using the linear time algorithm of Lee [16,17]. To compute the invisible region in which s (resp. t) lies (step 3), one 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 the vertices of $\mathcal{P}$ are in order. Step 5 is calling the shortest path algorithm of Guibas et al. [1] twice which can be done in $O(n)$ time. Since the funnel structure is stored in a finger search tree [18], which is based on the B-tree data structure, one 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 1. Each step of the algorithm uses, at most, $O(n)$ space, hence, the overall algorithm needs linear time and space. Thus, one has the following result for the case of simple polygons.

Theorem 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.

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. It is assumed that the domain is bounded by a given simple polygon with a number of (simple, non-overlapping) polygonal obstacles inside. By free space, one means 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 a simple polygon, one 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 where the only shortest p-visible path between s and t enters $\mathcal{V}_p$ from one edge and exits from another, while s and t both lie in one invisible region, W, and the only shortest path between them lies completely inside W (Figure 4).

Based on the above observation, the general approach presented in the previous section is used without major modifications for the case of polygonal domains. To compute L_s and L_t, the shortest path map of the domain is computed twice, with respect to the points s and t and the intersections of the two maps are found with $\mathcal{B}_p$. To find the intersections, one can use known algorithms for the subdivision overlay, such as the algorithm of Finke and Hinrichs [19], which

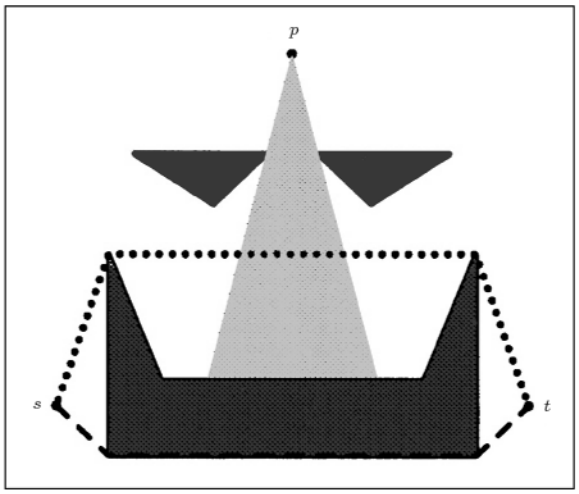

Figure 4. The shortest p-visible path between s and t (shown with dots) crosses part of $\mathcal{V}_p$ (lightshaded) while s and t are in one invisible region and the shortest path between them (dashed line) does not cross $\mathcal{V}_p$. Dark shaded polygons are obstacles.

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 $L_{s,t}$, one needs to merge the endpoints of the intervals in L_s and L_t, which requires having the intervals at each set in sorted order. The algorithm of Finke and Hinrichs produces the output subdivision in a quad view data structure [20], 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, the running time of the algorithm is analyzed in this case. Computing the visibility polygon can be done using the optimal algorithm of Heffernan and Mitchell [21], which requires $O(n + h\log h)$ time (h is the number of obstacles in the domain, which is $O(n)$ in the 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 1 holds in this case too. Thus, one has the following result for the case of polygonal domains.

Theorem 2

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.

POLYHEDRAL SURFACES

In this section, the geometric domain of polyhedral surfaces, are considered, i.e., $\mathcal{P}$ is the surface of a polyhedron. In the case of a convex polyhedron, $\mathcal{V}_p$ is a connected region whose boundary consists of $O(n)$ edges of $\mathcal{P}$. In a 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 and the algorithms for finding the visibility map are superquadratic in general [22], one assumes that the visibility region of the point to be seen is determined through a preprocessing stage and one focuses on finding a shortest p-visible path.

The problem of finding the shortest path between two points on the surface of a polyhedron is well studied. Especially, Chen and Han [5] presented a method for building a subdivision of the surface, which can be used for finding the 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 in [6], which finds a shortest path in $O(n\log^2 n)$, using the wavefront propagation method.

Shortest Paths on a Polyhedron

The related terminology borrowed from [4] is briefly reviewed. 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, $\mathcal{F} = (f_1, f_2, \cdots, f_{k+1})$, such that f_i is tedge-adjacent to f_{i+1} (sharing a common edge e_i). The (possibly empty) list of edges $\varepsilon = (e_1, e_2, \cdots, e_k)$ are referred to as an edge sequence and the vertex of face f_1 opposite e_1 is referred to as the root of ε (Figure 5).

Each face has a two-dimensional coordinate system associated with it. If faces f and f' are edge-adjacent sharing edge e, the planar unfolding of face f' onto face f is defined 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, it is said that an edge sequence $\varepsilon = (e_1, e_2, \cdots, e_k)$ is unfolded, 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 $(f_1, f_2, \cdots, f_k)$ lie in the plane of f_{k+l}.

It is said that a path π connects the edge sequence, $\varepsilon = (e_1, e_2, \cdots, e_k)$, if π consists of segments which join interior points of $e_1, e_2, \cdots, e_k$ (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 (taken from [4]) characterizes such paths.

Lemma 2

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.

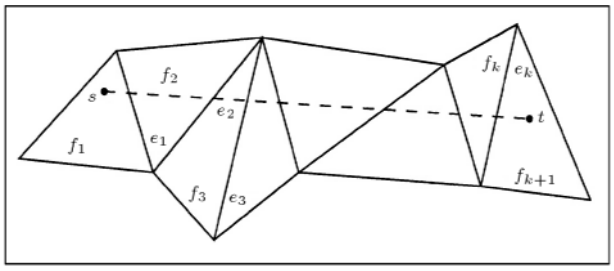

Figure 5. Shortest path from s to t unfolded along its edge sequence.

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 (Figure 6).

To compute the set, L_x, a subdivision presented in [5] is used, 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 that of a shortest path map in two dimensions. To obtain the subdivision, the surface of $\mathcal{P}$ is cut along the shortest paths from x to all 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 [23]). 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 the 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 (Figure 7). This subdivision has the property where 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 edge that are to be intersected with $\mathcal{B}_p$: The 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 the 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 cases. Note that it can be shown that these bounds are tight (Figure 8 shows a convex case).

For the non-convex case, the inward layout may overlap itself. The algorithm for computing the Voronoi diagram, in this case, is slightly different and is given in [5]. In this case, an interval in L_x has the property where there is a vertex, v, of the polyhedron, such that every shortest path from x to a point on the

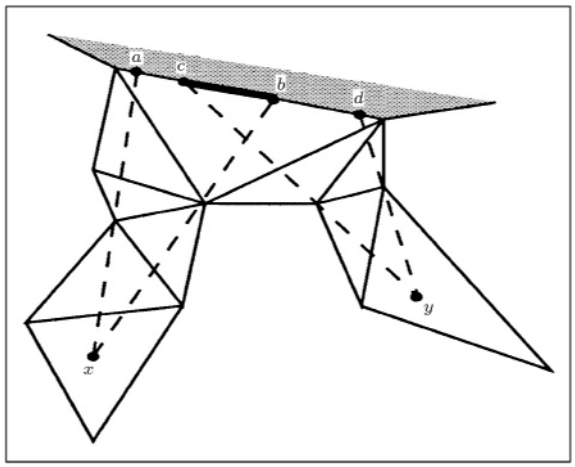

Figure 6. ab is an interval in L_x and cb is an interval in $L_{x,y}$.

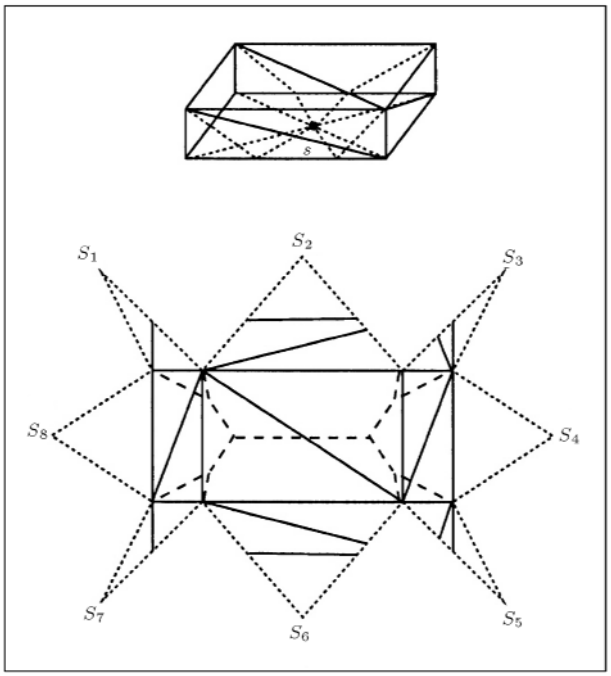

Figure 7. The inward layout of a box. Solid lines are the edges of the subdivision that are part of polyhedron edges. Dashed lines are edges of the Voronoi diagram (ridges) and dotted lines are shortest paths from images of the source to vertices (cuts).

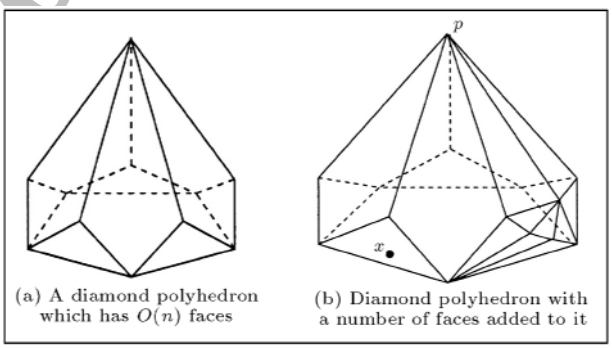

Figure 8. Constructing a convex polyhedron wiht $O(n^2)$ intervals.

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 the interval, given an interval, $I \in L_{s,t}$, one 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, the distance is stored to the 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, one 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 in 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 a total complexity of $O(n^2)$ edges (not necessarily parts of the edges of $\mathcal{P}$). In this case, one 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, one needs to intersect the edges of $\mathcal{B}_p$ with cuts and ridges, yielding to a 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, the following result is obtained for the case of polyhedral surfaces:

Theorem 3

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 $O(n^3)$ time for the non-convex case.

CONCLUSION

The problem of finding a shortest path between two points with single-point visibility constraint is studied 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 bound cannot be improved any further. However, the case of polygonal domains and polyhedral surfaces may be improved. 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 the existence of the subquadratic algorithm of Kapoor [6] for finding the shortest paths on polyhedral surfaces, which 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 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.

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	$O(n^3)$	$O(n^3)$

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

ACKNOWLEDGMENT

The authors would like to thank the anonymous referee for his/her valuable comments. Furthermore, this work has been supported by a grant from IPM School of Computer Science (No. CS 1382-2-02).

REFERENCES

1. Guibas, L.J. et al. "Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons", *Algorithmica*, **2**, pp 209-233 (1987).
2. Mitchell, J.S.B. "Shortest paths among obstacles in the plane", *Internat. J. Comput. Geom. Appl.*, **6**, pp 309-332 (1996).
3. Hershberger, J. and Suri, S. "An optimal algorithm for Euclidean shortest paths in the plane", *SIAM J. Comput.*, **28**(6), pp 2215-2256 (1999).
4. Mitchell, J.S.B. et al. "The discrete geodesic problem", *SIAM J. Comput.*, **16**, pp 647-668 (1987).
5. Chen, J. and Han, Y. "Shortest paths on a polyhedron", *Internat. J. Comput. Geom. Appl.*, **6**, pp 127-144 (1996).
6. Kapoor, S. "Efficient computation of geodesic shortest paths", *Proc. 32th Annu. ACM Sympos. Theory Comput.*, Atlanta, Georgia, USA, pp 770-779 (1999).
7. Boissonnat, J.D. et al. "Shortest paths of bounded curvature in the plane", *Internat. J. Intell. Syst.*, **10**, pp 1-16 (1994).
8. De Berg, M. and Van Kreveld, M. "Trekking in the alps without freezing or getting tired", *Algorithmica*, **18**, pp 306-323 (1997).
9. Efrat, A. et al. "Sweeping simple polygons with a chain of guards", *Proc. 11th ACM-SIAM Sympos. Discrete Algorithms*, San Francisco, California, USA, pp 927-936 (2000).
10. Khosravi, R. et al. "Shortest point-visible paths on polyhedral surfaces", *Proc. of the 10th International Conference on Computing and Information (ICC1'2000)*, Kuwait (2000).
11. Gudmundsson, J. and Levcopoulos, C. "Hardness result for TSP with neighborhoods", *Technical Report LU-CS-TR:2000-216*, Department of Computer Science, Lund University, Sweden (2000).
12. Mata, C. and Mitchell, J.S.B. "Approximation algorithms for geometric tour and network design problems", *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, Vancouver, BC, Canada, pp 360-369 (1995).
13. Gudmundsson, J. and Levcopoulos, C. "A fast approximation algorithm for TSP with neighborhoods", *Nordic J. of Computing*, **6**(4), pp 469-488 (1999).
14. Dumitrescu, A. and Mitchell, J.S.B. "Approximation algorithms for TSP with neighborhoods in the plane", *Proc. 12th A CM-SIAM Sympos. Discrete Algorithms*, Washington, DC, USA, pp 38-46 (2001).
15. Dror, M. et al. "Touring a sequence of polygons", *Proc. 35th ACM Sympos. Theory Comput.*, San Diego, CA, USA, pp 473-482 (2003).
16. Lee, D.T. "Visibility of a simple polygon", *Comput. Vision Graph. Image Process.*, **22**, pp 207-221 (1983).
17. Joe, B. and Simpson, R.B. "Correction to Lee's visibility polygon algorithm", *BIT*, **27**, pp 458-473 (1987).
18. Guibas, L.J. et al. "A new representation for linear lists", *Proc. 9th Annu. ACM Sympos. Theory Comput., Boulder*, Colorado, USA, pp 49-60 (1977).
19. Finke, U. and Hinrichs, K. "Overlaying simply connected planar subdivisions in linear time", *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, Vancouver, BC, Canada, pp 119-126 (1995).
20. Finke, U. and Hinrichs, K. "The quad view data structure: a representation for planar subdivisions", *Proc. 4th Sympos. Advances in Spatial Databases*, Portland, Maine, USA, pp 29-469 (1995).
21. Heffernan, P.J. and Mitchell, J.S.B. "An optimal algorithm for computing visibility in the plane", *SIAM J. Comput.*, **24**(1), pp 184-201 (1995).
22. O'Rourke, J. "Visibility", In *Handbook of Discrete and Computational Geometry*, J.E. Goodman and J. O'Rourke, Eds., pp 467-480, CRC Press LLC (1997).
23. Agarwal, P.K. et al. "Star unfolding of a polytope with applications", *SIAJ J. Comput.*, **26**(6), pp 1689-1713 (1997).