Computing Minimum Length Paths of a Given Homotopy Class

John Hershberger Jack Snoeyink

RUU-CS-90-37
December 1990

Utrecht University

Department of Computer Science

Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

ISSN:0924-3275

Computing minimum length paths of a given homotopy class[*]

John Hershberger
DEC Systems Research Center
130 Lytton Avenue
Palo Alto, CA 94301

Jack Snoeyink
Department of Computer Science
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
the Netherlands[†]

December 21, 1990

Abstract

In this paper, we use the universal covering space of a surface to generalize previous results on computing paths in a simple polygon. We look at optimizing paths among obstacles in the plane under the Euclidean and link metrics and polygonal convex distance functions. The universal cover is a unifying framework that reveals connections between minimum paths under these three distance functions, as well as yielding simpler linear-time algorithms for shortest path trees and minimum link paths in simple polygons.

1 Introduction

If a wire, a pipe, or a robot must traverse a path among obstacles in the plane, then one might ask what is the best route to take. For the wire, perhaps the shortest distance is best; for the pipe, perhaps the fewest straight-line segments. For the robot, either might be best depending on the relative costs of turning and moving.

In this paper, we find shortest paths and closed curves that wind around the obstacles in a prescribed fashion—that have a certain homotopy type. We consider the Euclidean and link metrics for general paths, and convex and link distance functions for paths restricted to use c given orientations, such as rectilinear paths. Our work presents these distance functions in a unified framework and generalizes previous results for paths in simple polygons. It also results in simplified linear-time algorithms for Euclidean shortest path trees and minimum link paths and new algorithms for c-oriented paths in simple polygons.

[*]This research was supported by Digital Equipment Corportation and the ESPRIT Basic Research Action No. 3075 (project ALCOM)

[†]On leave from the Department of Computer Science of the University of British Columbia. Portions of this research was performed while at the Computer Science Department of Stanford University.

In the remainder of this section, we review previous results on finding minimum Euclidean and link paths, closed curves, and c-oriented paths. In section 2, we define triangulated manifolds, homotopy classes, and the universal covering space—important topological tools for our algorithms. Section 3 investigates general paths under the Euclidean metric. It gives algorithms for Euclidean shortest paths of a certain homotopy class, shortest closed curves, and shortest path trees. Section 4 deals with the link metric, in which the length of a path is the number of its line segments. It gives algorithms for minimum link paths and discusses closed curves. Finally, section 5 develops similar algorithms for paths restricted to c previously chosen orientations, such as rectilinear paths.

1.1 Euclidean shortest paths

Many researchers have investigated the problem of finding minimum Euclidean paths in simple polygons. Chazelle [6] and Lee and Preparata [28] gave algorithms that, in triangulated polygons, compute the shortest Euclidean path between two points in linear time. We extend their ideas to universal covers and also to maintaining the shortest path homotopic to a path given on-line. Guibas et al. [20] showed how to compute the tree of all shortest paths from one polygon vertex to all other vertices in linear time; they use this as a preprocessing step to solve several shortest path and visibility query problems. One application of our on-line path algorithm is to compute the shortest path tree using simpler data structures.

Shortest Euclidean paths have been studied as one of the tractable cases of river routing in VLSI. Cole and Siegel [9], Leiserson and Maley [29], and Gao et al. [16] give algorithms for routing wires with fixed terminals among fixed obstacles when a *sketch* of the wires is given—that is, when a homotopy class is specified for each wire. When no sketch is given or when the terminals are not fixed, the resulting problems are usually *NP*-hard [30, 34, 37]. Both Leiserson and Maley and Gao et al. use an algorithm much like our Euclidean minimum path algorithm to compute the *rubber-band equivalent* of each wire as a basic preprocessing step.

Finding minimum paths among obstacles when the homotopy class is not given is a more difficult problem, and one that we will not discuss. For the Euclidean metric, one typically builds the visibility graph and searches it with Dijkstra's algorithm [14]; see Ghosh and Mount [18] and Kapoor and Maheshwari [26] for efficient algorithms.

There are special closed curves of interest to computational geometers that fit within the framework of this research. Under the Euclidean metric, the minimum length enclosing curve of a set of points or line segments is the convex hull of the set. The minimum length curve enclosing a set and contained in the interior of a polygon is the *relative convex hull* of the set. Relative convex hulls, also called geodesic hulls, have been studied by Toussaint [44], who gave an $O((m+n)\log(m+n))$ time algorithm for computing the relative convex hull of m points in an n-gon and a linear time algorithm based on a triangulation for computing the relative convex hull of one simple polygon with respect to another. Our algorithms run in the same time. Guibas and Hershberger [24] gave an $O(n + m\log(n+m))$ time algorithm as an application of two-point shortest path queries.

Relative convex hulls are studied in connection with the separability of simple polygons under translation [4, 11, 42, 43].

1.2 Link shortest paths

Researchers have also looked at finding minimum paths in simple polygons under the link metric, in which the length of a path is the number of its line segments. Suri [41] developed a linear time algorithm for computing the minimum path between two points in a simple polygon. Ghosh [17] recently gave a linear time algorithm as a consequence of his work on computing the visibility polygon from a convex set. Both algorithms are based on a triangulation and the shortest path tree algorithm of Guibas et al. [20]. We show how to extend our Euclidean minimum path algorithm to compute the link minimum path in time proportional to the number of triangles the path intersects. This gives yet another linear-time algorithm in a simple polygon, but one that is more direct and also has application to paths of given homotopy class among obstacles.

When the homotopy type is not specified, Mitchell, Rote, and Woeginger [31] have given an algorithm that runs in $O(E\alpha(n)\log^2 n)$, where n is the number of vertices, E is the size of the visibility graph, and $\alpha(n)$ is the inverse of Ackermann's function.

Minimum link curves enclosing a set and contained in a polygon separate the set and polygon using the smallest number of line segments. Aggarwal et al. [1] considered finding a minimum link convex polygon separating two convex polygons with n total vertices. They obtained an $O(n \log k)$ algorithm that finds the minimum polygon of k line segments. They also give a simple $O(n)$ algorithm for finding a polygon with at most one segment more than the minimum. Wang and Chen [45] show that the algorithm of Aggarwal et al. can find the minimum link convex polygon that encloses a convex polygon lying in the kernel of a star-shaped polygon. They reduce two polygons with a total of n vertices to this case in $O(n \log n)$ time. Ghosh [17] computes the reduction in linear time, allowing the computation of the minimum link convex separator in $O(n \log k)$ time. For non-convex polygons, Suri and O'Rourke [40] compute a minimum link polygon separating an m-gon and its enclosing n-gon in $O(mn)$ time. We use our minimum link path algorithm to find a non-convex polygon in linear time after triangulation and use Ghosh's refinement of Aggarwal et al.'s algorithm if the separating polygon is convex.

1.3 Paths restricted to c orientations

In some applications, most notably VLSI, the orientations of paths are restricted. Rectilinear paths are the most important and, thus, the most studied.

For computing rectilinear shortest paths among rectilinear barriers under the Manhattan, or L_1, metric, researchers have developed algorithms that work in simple polygons (e.g. [38]) and in the presence of obstacles [8, 13, 27]. The fastest algorithms for the (globally) shortest path among rectilinear obstacles have subquadratic worst-case complexity: $O(n \log n)$ if the obstacles are disjoint [13] and $O(n \log^2 n)$ if they are not [8].

Mark de Berg [10] has given an algorithm that finds a path that is both a minimum link and an L_1 shortest path in a simple polygon. He and others [12] give a quadratic algorithm for a combined link and L_1 metric for paths among obstacles.

Güting [25] defined c-oriented polygons as a generalization of rectangles; he and others, such as Rawlins and Wood [36], have looked at geometry problems with restricted orientations. We, however, seem to be the first to present algorithms for c-oriented paths in a simple polygon.

2 Preliminaries

We begin by defining some important topological objects.

2.1 Manifolds, complexes, and metrics

Our results apply to *boundary-triangulated 2-manifolds* (BTMs), which we define below. BTMs are slightly more general than polygonal regions in the Euclidean plane. We consider them primarily because every BTM has a simply-connected covering BTM such that paths have a unique lift into the covering space. More on this later, however.

First, recall that a two-dimensional manifold with boundary (a *2-manifold*) is a topological space in which each point has an open neighborhood homeomorphic to a two-dimensional ball or half-ball. The former are *interior points* and the latter are *boundary points*. We restrict our attention to orientable manifolds—no Möbius bands.

A two-dimensional *simplicial complex* is a triangulated 2-manifold. Spelled out, a two-dimensional simplicial complex is a collection of triangles, edges, and vertices such that any two triangles either do not intersect, intersect at a vertex, or intersect at two vertices and their common edge; no other intersections are permitted. At most two triangles are incident to an edge; edges incident to a single triangle are *boundary edges*. Furthermore, all the triangles and edges incident to a vertex can be ordered so that adjacent edges and triangles are coincident. All vertices are either *boundary vertices* with two incident boundary edges, or *interior vertices* with none.

Finally, a *boundary-triangulated 2-manifold* or BTM is a simplicial complex in which all vertices are boundary vertices. Figure 1 depicts two simplicial complexes; the second is a BTM.

Simplicial complexes can be represented by Guibas and Stolfi's quad-edge structure [23], by Baumgart's winged-edge structure [3], or by their dual graph. We require that each triangle be able to access its edges and each edge its incident triangles in constant time. If a polygonal region is given, we can triangulate it in $O(n \log n)$ time by a sweepline algorithm [35] or, if there are a constant number of boundary components, in linear time [7].

We assume that a path α is given in some form that can be traced through the data structure that represents a BTM M in time proportional to α's complexity, C_α, plus the number of times α crosses a triangulation edge, Δ_α. If α is piecewise linear, then C_α is the number of α's line segments and, using any of the previous data structures, we can compute a constant number of segment/segment intersection points and determine the

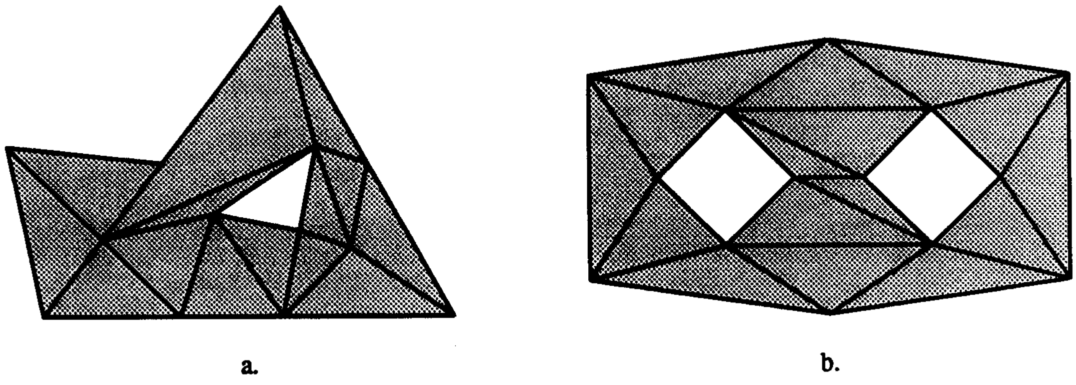


Figure 1: Two simplicial complexes; one is a boundary-triangulated 2-manifold (BTM)

next triangulation edge that would be hit if the path leaves the current triangle. Thus we traverse α in $O(C_\alpha + \Delta_\alpha)$ time.

We consider two metrics for unrestricted paths in a BTM: the Euclidean and link metrics. The *Euclidean* metric is the usual L_2 metric; the length of a curve or subdivision is the sum of the lengths of its pieces in all the triangles it intersects. In the *link* metric, the length of a curve or subdivision is the number of its line segments. Under both metrics, the minimum length paths are composed of line segments.

In applying the link metric, we would like to consider two adjacent triangles of a BTM to be coplanar, even if they are not. Thus, contiguous "line segments" that would be collinear if the triangles they passed through were laid out flat in the plane are counted as a single segment. This unfolding process is what is used to find shortest paths on the surface of a polyhedron [32, 39].

For some applications, such as VLSI, the paths constructed must use a constant number of fixed directions or orientations. Rectilinear paths with the four orientations of north, south, east and west are the most common. When paths are restricted, the link metric remains the number of line segment of a path. The Euclidean metric, however, should be replaced by a distance function that gives the length of the shortest restricted path between two points. We discuss this more fully in section 5.

A useful example of a BTM is a triangulated polygonal region R in the Euclidean plane: a set bounded by n line segments with disjoint interiors. If one considers the line segments as obstacles and looks at paths avoiding the obstacles, then one can form equivalence classes of paths by relating paths that can be deformed to each other within R.

2.2 Homotopy classes

The topological concept of *homotopy* formally captures the notion of deforming paths and curves. Let α and β be functions from a topological space X to a topological space Y that are continuous; that is, the preimage $\alpha^{-1}(A)$ of an open set $A \subseteq Y$ is open. Functions α and β are *homotopic* if there is a continuous function $\Gamma: X \times [0,1] \to Y$ such that $\Gamma(x,0) = \alpha(x)$ and $\Gamma(x,1) = \beta(x)$. One can see that homotopy is an equivalence relation [2, 33]. In this paper, the range set Y is always a boundary-triangulated 2-manifold

M under the subspace topology. We specify the set X in two different ways.

First, we consider paths joining two given points, p and q: a *path* in M is the range of a continuous function $\alpha: [0,1] \to M$. We set $X = [0,1]$ and require that the endpoints of a path α be fixed at $\alpha(0) = p$ and $\alpha(1) = q$. Two paths are *path homotopic* if one can be deformed to the other in M while keeping the endpoints fixed. Formally, paths α and β are *path homotopic* if there is a continuous map $\Gamma: [0,1] \times [0,1] \to M$ such that $\Gamma(x,0) = \alpha(x)$ and $\Gamma(x,1) = \beta(x)$, and $\Gamma(0,y) = \alpha(0) = \beta(0)$ and $\Gamma(1,y) = \alpha(1) = \beta(1)$.

Second, a closed curve is the image of a circle under a continuous map into M. Thus, we set $X = S^1$: the unit circle under the standard topology. Two curves with maps α and β are *homotopic* if there is a continuous map $\Gamma: S^1 \times [0,1] \to M$ such that $\Gamma(x,0) = \alpha(x)$ and $\Gamma(x,1) = \beta(x)$.

One can go on to define define homotopy in M for two subdivisions α and β—indeed, we do so in a paper with Leonidas Guibas and Joseph Mitchell [22] and show that computing minimum link subdivisions is *NP*-hard. But that does not fit into our framework of universal covering spaces, which we will define in the next section.

A homotopy relation partitions paths or closed curves into equivalence classes. Thus, we can describe a homotopy class by giving a representative path, or closed curve, α. Given α, we seek to compute a minimum length representative of α's class under the Euclidean and link metrics.

Let's look at one concrete example of a path homotopy. In a BTM, a path gives a sequence of triangles and triangulation edges; we can form a *canonical path* that visits the midpoints of the triangles and edges in the same sequence. It is easy to see that a path is homotopic to its corresponding canonical path—at times we will find it convenient to use the canonical path as the representative of a homotopy class.

One can concatenate two paths if one ends where the other begins. The next theorem is a well-known tool for studying paths.

Theorem 2.1 ([2, 33]) *The operation of path concatenation has group properties: associativity, identities, and inverses.*

This has an easy corollary for simply-connected 2-manifolds, in which every loop is homotopic to a point.

Corollary 2.2 *In a simply connected manifold, any two paths with the same starting and ending point are homotopic.*

This corollary becomes useful because we can always find a simply-connected covering space, which we now define.

2.3 Covering spaces

Informally, a topological space U is a covering space of a space X if, at each point $u \in U$, there is a corresponding point $x \in X$ such that things around u and x look the same in their respective spaces, but there may be many points of U mapping to the same point x.

Formally, let $p: U \to X$ be a continuous and onto map between connected topological spaces U and X. If every point $x \in X$ has an open neighborhood N where the inverse image $p^{-1}(N)$ is a union $\bigcup_i U_i$ of disjoint open sets of U and the restriction $p|_{U_i}$ is a homeomorphism from U_i onto N, then p is a covering map and U is a covering space of X.

A space is always a covering space of itself under the identity map. For a more useful example, consider the covering space of a BTM M formed by the following procedure (see figure 2): Choose a base triangle of M, copy it, and make its edges active. Now, any triangle t with an active edge e is a copy of some triangle $t' \in M$ and of an edge e' of t'. There is another triangle $u' \in M$ incident to e'—copy it, forming u, and attach u to t along edge e. Make edge e inactive and the other two edges of u active. One can see that the function that sends the copy of a point to its original is a covering map. The covering space thus formed is the *universal covering space* of M.

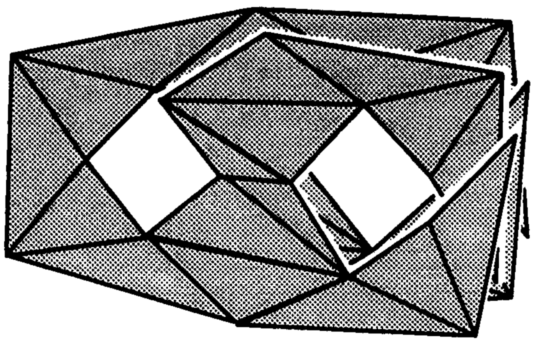


Figure 2: A portion of the universal cover

The dual graph of the universal covering space, the graph with a node for each triangle and an arc joining nodes that correspond to triangles that are incident to the same edge, is an infinite tree rooted at a copy of the base triangle. One can show that the dual graph, considered as an unrooted tree, is not affected by the base triangle chosen, so the universal cover does not depend on the base triangle. Furthermore, the universal cover is simply connected—it has no holes:

Lemma 2.1 *The universal covering space U of a BTM is simply connected.*

Proof: Consider any path α starting and ending at a point p in the universal covering space U of the BTM M. The path α intersects some connected subset of the triangles of the covering space.

If α intersects only one triangle, then α collapses to the point p by the homotopy $f(t) = (1-t)\alpha + tp$. Otherwise, we can consider the triangle containing p as the base triangle for the covering space. The dual graph of the space is a tree, so the dual of the connected subset of triangles that α intersects must also be a tree. In any leaf, loops of α start and end at the same edge; we can deform these loops to the edge by an easy homotopy and trim the leaves. By induction, α can be contracted to p.

Thus, the universal covering space is simply connected. ∎

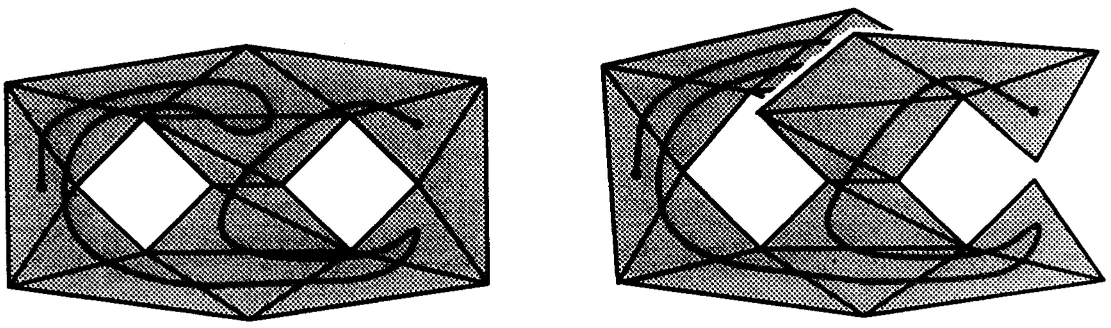


Figure 3: The lift of α

Any path that begins in the base triangle has a unique lifting to the covering space, as indicated in figure 3. Formally, let $p: U \to M$ be a covering map. If a function f, from a space W to the BTM M is one-to-one and continuous, then a *lifting* of f is a map $\hat{f}: W \to U$ such that the composition $p\hat{f} = f$. When we lift a path α, we use $U_\alpha \subset U$ to denote the BTM composed of the triangles of the universal cover U that intersect the path $\hat{\alpha}$.

One last lemma pulls all of the constructs in this section together:

Lemma 2.2 *If α is a path in a BTM, M, then we can construct U_α, the portion of the universal covering space of M that contains the lift of α, in $O(C_\alpha + \Delta_\alpha)$ time.*

Proof: The construction algorithm is simple: Begin with U_α equal to a copy of the triangle of M that contains the starting point of the path α. Then trace α through M and, simultaneously, the lift of α through the covering space—when α crosses a triangulation edge into a triangle of M, add a copy of the triangle to U_α if the lift has never crossed the corresponding edge before. (Otherwise, the triangle is already present.) We can trace the path α through the triangles of M in the stated time bound. ∎

3 The Euclidean metric

We investigate Euclidean shortest paths and closed curves in three subsections. In section 3.1, we remark how the algorithm of Chazelle [6] and Lee and Preparata [28] can be used to find shortest paths between two points of a given homotopy type. Section 3.2 modifies the algorithm to compute shortest closed curves, which include the relative convex hull. Then, in section 3.3, we maintain the shortest path on-line and, as a by-product, find shortest path trees in linear time without using finger search trees. This simplifies an important algorithm of Guibas et al. [20].

3.1 Funnels and the shortest path between two points

First we review funnels, a data structure used by many researchers to find shortest paths [6, 16, 20, 24, 28, 29].

Let p be a point and $\overline{uv}$ be a line segment in a simply connected BTM. The shortest paths from p to v and from p to u may travel together for a while. At some point a they diverge and are concave until they reach u and v, as illustrated in figure 4. The region bounded by $\overline{uv}$ and the concave chains to a is called the *funnel;* a is the *apex* of the funnel.

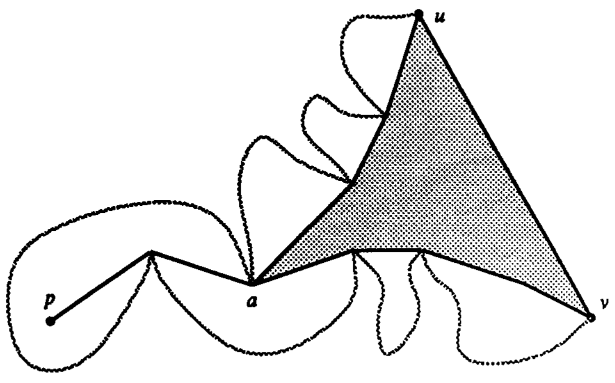


Figure 4: A funnel

We store the vertices of a funnel in a double-ended queue, a *deque.* Figure 5 shows that the extensions of funnel edges define wedges. When a point w is added to one end of the funnel, we pop points from the deque until we reach b, the apex of the wedge that contains w, then we push w. If the apex of the funnel is popped during the process, then b becomes the new funnel apex. Notice that the edge $\overline{bw}$ is on the shortest path to w.

The shortest path algorithms of Chazelle [6] and Lee and Preparata [28] both look for a path in a *sleeve* polygon—a triangulated simple polygon whose dual tree is simple path. We shall look for a *sleeve* BTM.

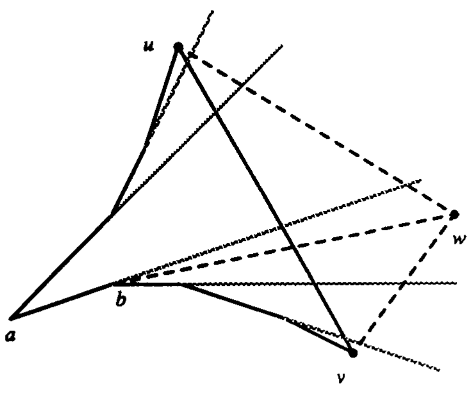


Figure 5: Splitting a funnel about w

Lemma 3.1 *Let α be a path from p to q. One can compute, in $O(C_\alpha + \Delta_\alpha)$ time and space, a sleeve BTM that contains the Euclidean shortest path homotopic to α.*

Proof: Choose the triangle that contains p as the base triangle and construct U_α, the portion of the universal cover that contains the lift of α, according to lemma 2.2.

In the dual tree of U_α, there is a unique path to the triangle containing the lift of q; let α' be the canonical path in U_α that corresponds to this dual path. Since U_α is simply connected, paths α and α' are homotopic (corollary 2.2).

The BTM $U_{\alpha'} \subseteq U_\alpha$ is a sleeve. A boundary edge e of $U_{\alpha'}$ may separate the universal cover but can not separate p from q. Any path homotopic to α' that crosses e does so twice and can be shortened by following e. Thus, the shortest path from p to q homotopic to α' (and to α) is contained in $U_{\alpha'}$. ∎

Trace the canonical path α' through $U_{\alpha'}$ and maintain the funnel of the triangulation edges crossed. The set of all edges added to the funnel comprises the shortest path tree rooted at p, that is, the union of all shortest paths from p to vertices of $U_{\alpha'}$. From this tree it is easy to recover the shortest path from p to q. Thus, we have obtained

Theorem 3.1 *The Euclidean shortest path that is homotopic to a given path α can be computed in $O(C_\alpha + \Delta_\alpha)$ time and $|U_\alpha|$ space.*

3.2 Shortest closed curves

The algorithm of the previous section can be extended, without much difficulty, to compute Euclidean minimum closed curves of a given homotopy type. The running time, once the space has been triangulated, is linear in the number of triangles explored. This matches the time bounds of algorithms that compute relative convex hulls [11, 42, 44], the most practical instance of the minimum closed curve problem.

We begin by building an annulus that contains the shortest curve.

Lemma 3.2 *In $O(C_\alpha + \Delta_\alpha)$ time, we can build a BTM homotopic to an annulus that contains the shortest path homotopic to a closed curve α.*

Proof: Choose any point p on α and find the sleeve of the path from p to p along α. An initial sequence of triangles and triangulation edges of this sleeve will appear in reverse order at the end of the sleeve (unless the sleeve consists of only the triangle containing the point p). We remove all but the last of these common triangles, which we glue together. Because the manifold was assumed to be orientable, the result is an annulus. Cutting along one of the non-boundary edges of the annulus makes it simply connected; any curve homotopic to α must intersect all these edges. A curve that crosses a boundary edge e does so twice and can be shortened by following e. Thus, the shortest curve is contained in the annulus. ∎

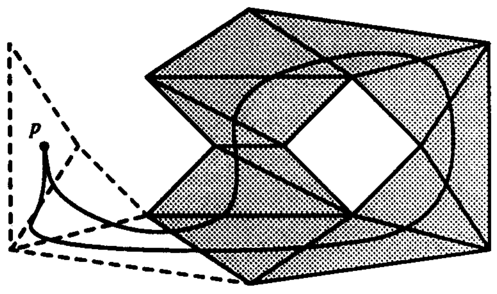


Figure 6: The annulus

We can find the shortest path around the annulus by following the canonical path around until we meet the shortest path. We prove:

Theorem 3.2 *One can compute the shortest curve around an annulus in time proportional to the number of its triangles.*

Proof: Choose a point p in a triangle t and follow the canonical path around the annulus, maintaining the shortest path from p. When the canonical path returns to t, then switch to following the shortest path—still maintaining the shortest path from p.

When the current path shares an edge $\overline{st}$ with the shortest path from p, then stop. The shortest paths from t to t and from s to s are co-incident, so they must be the shortest closed curve. ∎

```
Length(l)                          Index(l, i)
    return last − first + 1            check 0 ≤ i < Length(l)
                                       return l[i + first]
Add(f, l, x)
    decrement first                Undo()
    push (Add, f, l[first]) to stack   if stack top is (Add, f, x)
    set l[first] ← x                       set l[first] ← x
                                           increment first
Split(f, l, i)                         else stack top is (Split, f, i)
    check 0 ≤ i < Length(l)                set last ← i
    push (Split, f, last) to stack
    set last ← i
```

Table 1: Code fragments for the front-of-list operations

3.3 On-line shortest paths and shortest path trees

In this section, we show how to trace the lift of the path α from p to q and maintain the funnel used in shortest path computations. This procedure optimizes a path that is given on-line and is useful for interactive applications.

We can apply this procedure to compute shortest path trees in a simply connected BTM; the *shortest path tree* from a point p is the union of all shortest paths from p to vertices of the BTM. Guibas et al. [20] compute the shortest path tree of any triangulated simple polygon by splitting funnels—they use finger search trees to find the splitting vertices efficiently. We find the shortest path tree by tracing the boundary and maintaining the funnel; the edges added to the funnel compose the tree. Our algorithm runs in the same time and uses arrays in place of finger search trees.

First we describe a data structure that supports five operations on ordered lists:

Length(l)	Return the number of items in the list.
Index(l, i)	Return the ith item in the list.
Add(f, l, x)	Add the item x to the f=front (or b=back) of the list l.
Split(f, l, i)	Return the sublist of l in f=front (or b=back) of and including item i and discard the other half of the list.
Undo()	Undo the most recent **Add**() or **Split**() operation.

We store the list in the entries of an array with indices from *first* through *last*. When we perform an **Add**() or **Split**(), we record the changed array entries or indices in a history stack so that the **Undo**() operation can return the array to the previous state. The code fragments in table 1 indicate that the operations can be implemented to run in constant time. If we begin with an empty list, denoted by indices *first* $= n$ and *last* $= n - 1$, and perform at most n **Add**() operations, then an array of size $2n$ is sufficient.

Using this data structure, we can trace the lift of the path α from p to q and maintain a list that represents the funnel defined by the shortest paths from p to the current BTM

```
Initialize
    list ← {$p$}
    stack empty
    begin the trace of $\alpha$ at $p$
repeat
    if the trace visits a new triangle $\triangle$
        add $\triangle$ to $U_\alpha$
        perform increasing increment search
            for the split index $i_\triangle$
    fi
    trace $\alpha$ to next edge $\overline{uv}$, with $u$ left of $v$
    if crossing out of base$\triangle$:
        Add(f, list, $u$)
        Add(b, list, $v$)
    else if crossing away from base$\triangle$:
        if $u$ is in the funnel, $v$ is new
            Split(b, list, $i_\triangle$)
            Add(f, list,v)
        else if $u$ is in the funnel, $v$ is new
            Split(f, list, $i_\triangle$)
            Add(b, list,u)
        fi  else if crossing toward base$\triangle$:
        Undo(list)
        Undo(list)
    fi
    trace $\alpha$ into next triangle
until done tracing
```

Table 2: Outline of the funnel maintenance algorithm

edge. Table 2 gives an outline of the algorithm; we describe it below.

We begin in the base triangle with a funnel list containing only the point p. (Whenever the lift of α is in the base triangle, the funnel is the point p.) When α crosses an edge out of the base triangle, we add the edge's endpoints to the funnel.

Suppose the path α crosses an edge $\overline{uv}$ into a new triangle $\triangle uvw$ of U_α. If α leaves through one of the other edges of $\triangle uvw$, then the current funnel list is split into the funnel defined by $\overline{uw}$ or by $\overline{vw}$ as illustrated in figure 5. We compute the index i where the list splits when the point w is added and store i with the triangle $\triangle uvw$. We use an increasing increment search from the front of the list to find the index i in $2\log i$ probes: check the extension of the 1st, 2nd, 4th, 8th, etc, edge of the funnel until we pass the point w, then perform binary search to find the wedge containing w. By searching from the front and back simultaneously, we find the splitting index in $O(\log d)$ steps, where $d = \min\{i, \mathbf{Length}(l) - i\}$. Finger search trees were used in [20] to implement the simultaneous increasing-increment search, but arrays avoid the extra pointer complexity.

Consider the dual graph of U_α—the triangles of the universal cover that intersect the lift of α—as a tree rooted at and directed toward the base triangle. When the path α encounters a triangulation edge, α is heading either away from or toward the base triangle. If α is heading away, then we perform a **Split**() indicated by the index stored in the current triangle and **Add**() the new triangle vertex to obtain the next funnel. If α is heading toward the base, then we undo the last split/add pair. Lemma 3.3 establishes an important invariant of the algorithm and implies that, in the first case, the correct split index is stored and that, in the second case, the correct split/add pair is undone.

Lemma 3.3 *Suppose the algorithm has traced a path α from p up to q. The pairs of funnel operations on the history stack come from the triangulation edges crossed in order by the shortest path from p to q homotopic to α.*

Proof: We prove this lemma by induction on the number of triangulation edges that α crosses. The invariant holds trivially if α is entirely contained in the base triangle.

Suppose the invariant holds for all paths crossing j triangulation edges. Let α be a path composed of a path α' crossing j triangulation edges and a path α'' crossing one edge. By the induction hypothesis, the operations used to construct the funnel of the path α' are those of the shortest path homotopic to α' in order. The operations that advance from one triangle to the next, however, preserve this invariant. ■

Except for finding the splitting index—which one does once for each triangle of the universal cover U_α—one does a constant amount of work when visiting a triangle. The recurrence relation analysis of [20] shows that the time to find the splitting indices is also linear in Δ_α. Thus, we have established the following theorem.

Theorem 3.3 *One can trace a path α through the universal cover of a BTM and maintain the funnel in $O(C_\alpha + \Delta_\alpha)$ time and space.*

If P is a triangulated simple polygon and α is the path from a vertex p around the boundary of P and back to p, then the algorithm computes the edges of the shortest paths from p to vertices of the polygon—that is, the shortest path tree of P.

4 The link metric

In the link metric, the length of a path or closed curve is the number of its line segments. Section 4.1 gives an algorithm to compute a minimum link path of a homotopy class in time proportional to the number of triangles that it intersects and Section 4.2 briefly discusses minimum link closed curves.

4.1 Computing the minimum link path

In this section we show how to compute the minimum link path, $\alpha_{\min}$, homotopic to a given path α in time proportional to $C_\alpha + \Delta_\alpha + \Delta_{\alpha_{\min}}$, the complexity of the path α plus the number of triangles intersected by both paths. Our approach is inspired by Ghosh's [17] observations about the relationship between minimum Euclidean and link paths in simple polygons. We compute the Euclidean shortest path and then use a greedy approach to minimize the number of line segments. Our algorithm is output-sensitive and is simpler than that of Ghosh in the simple polygon case; it avoids his middle step of computing a visibility polygon.

First, some definitions: Since we have enough time to compute the Euclidean shortest path in the homotopy class of α, we may assume that α is the shortest path from p to q. As before, U is the universal covering space and U_α consists of the triangles of U that intersect α.

Traversing α from p to q, we can label each vertex as a *left* or *right* turn. We call an edge $\overline{tu}$ of α an *inflection edge* if the labels of t and u differ; edges incident to p and to q can also be called inflection edges. (Ghosh calls such edges eaves.) The extension of a line segment $\overline{tu}$ in U, denoted $ext(\overline{tu})$, is the line segment, ray, or infinite line formed by extending $\overline{tu}$ until it hits boundary points of U. In a simple polygon, Ghosh observed that there is always a minimum link path including one line segment from the extension of each inflection edge. This is also true in a simply connected BTM:

Lemma 4.1 *If $\overline{tu}$ is an inflection edge of a Euclidean shortest path α, then a minimum link path homotopic to α can be assumed to use a subsegment of $ext(\overline{tu})$.*

Proof: Let s and v be the endpoints of the extension $ext(\overline{tu})$ so that these points appear in order s, t, u, v. Each of the segments $\overline{st}$, $\overline{tu}$, and $\overline{uv}$ separates p from q in the universal covering space U, so any path from p to q must cross all three segments, as shown in figure 7. If a path α' from p to q intersects $\overline{st}$ at a and $\overline{uv}$ at b, we can shortcut α' with the segment $\overline{ab} \subseteq ext(\overline{tu})$. Since some line segment of α' intersected $\overline{tu}$, this shortcut does not increase the number of segments on the path. ∎

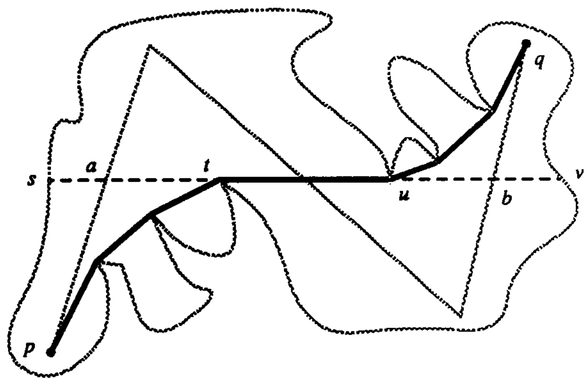


Figure 7: Extension $ext(\overline{tu})$ separates U

Thus we can assume that any inflection edges are included in the minimum link path.

We have reduced our problem to one of finding the minimal link path from $\overline{uv}$, a segment extending one inflection edge, to $\overline{u'v'}$, a segment extending another, where the shortest path from u to u' is concave; see figure 8.

If the extension segments $\overline{uv}$ and $\overline{u'v'}$ intersect in U_α, then no additional segments are needed. Otherwise, consider the Euclidean shortest path γ from v to v' in U_α; the path from u to u' and γ form what has been called the *hourglass* of $\overline{uv}$ and $\overline{u'v'}$. The path γ helps find a segment of the minimum link path.

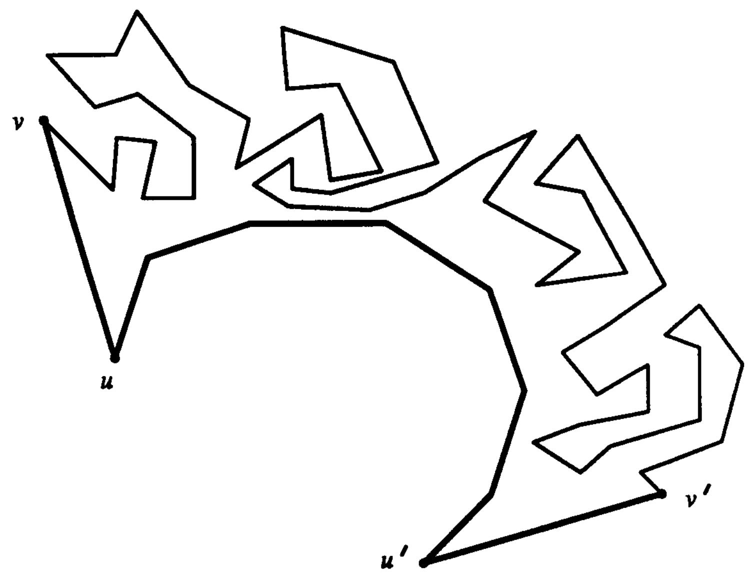


Figure 8: The reduced problem

Lemma 4.2 *In the polygon pictured in figure 8, the minimum link path joining $\overline{uv}$ and $\overline{u'v'}$ either has one segment or can be chosen to include an inflection edge of γ, the shortest path from v to v'.*

Proof: If γ is concave, then the concave chains can be separated by a line; one segment can join $\overline{uv}$ to $\overline{u'v'}$.

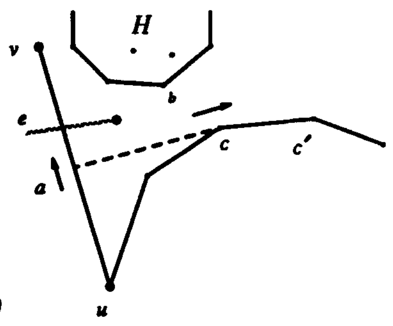
Otherwise, γ has an inflection edge. Let $\overline{bc}$ be the inflection edge closest to v as shown in figure 9. (We consider v to be labeled opposite u so that b may be v.) Because the paths from u to c and v to c are both concave, the extension of $\overline{bc}$ intersects $\overline{uv}$ at some point a. Let $\overline{cd}$ be the extension of $\overline{bc}$ through c in U_α. Any path from $\overline{uv}$ to $\overline{u'v'}$ must intersect both $\overline{bc}$ and $\overline{cd}$, so shortcutting the path with a subsegment of $\overline{ad}$ does not increase the number of its line segments. ∎

Finally, we discover the inflection edge $\overline{bc}$, if it exists, in time proportional to the number of triangles that $\overline{ac}$ intersects by the procedure outlined in table 3 and described in the rest of this section.

Notice that $\overline{ad}$ is tangent to the concave chain. We find $\overline{ad}$ by moving the point a up the edge $\overline{uv}$ and maintaining the point c tangent to the chain. We stop the motion when one of three cases occurs:

1. The tangent $\overline{\dot{a}\dot{c}}$ becomes the extension of $\overline{u'v'}$: no extra segments are needed.
2. The moving point a reaches v: the segment $\overline{vc}$ is the inflection edge.
3. The tangent $\overline{\dot{a}\dot{c}}$ encounters a point b between a and c: the segment $\overline{bc}$ is the inflection edge.

The third case is the most difficult to detect; we use the following technical lemma:



```
Variables:
    Points $a$, $c$, $c'$:
        $a$ sweeps up $\overline{uv}$
        $\overline{ac}$ tangent to concave chain at $c$
        $c'$ is point on concave chain following $c$
    Edge $e$
        first triangulation edge (t-edge) hit by $\overrightarrow{av}$
Data structure:
    $(H, b_\theta)$: Convex hull and point with tangent of slope $\theta$
    Description:
        Uses Graham scan [19] to maintain convex hull of
        endpoints of triangulation edges (t-edges)
    Operations:
        Add($p$, f or b): Add points to front or back of $H$
        ChangeSlope($\theta$): Change slope and recalculate $b_\theta$
Initialize
    $a \leftarrow u$, $c \leftarrow next(a)$, and $c' \leftarrow next(c)$
    $e \leftarrow$ first t-edge hit by $\overrightarrow{av}$
    for each t-edge crossing $\overline{ac}$ in order from $a$ to $c$
        if an endpoint $p$ lies in $\angle vac$ then Add($p$, b)
repeat
    ChangeSlope(slope of $\overline{ac}$)
    move $a$ along $\overline{uv}$, rotating around $c$ until
        case 1: $c = u'$ and $a$ reaches $\overline{u'v'}$
            use $\overline{au'}$ in minimum link path
            exit program
        case 2: $a$ reaches $v$.
            use $\overline{vc}$ in minimum link path
            exit loop
        case 3: Line segment $\overline{ac}$ hits $b_\theta$
            use $\overline{ac}$ in minimum link path
            exit loop
        case 4: slope of $\overline{ac}$ points into $H$ at $b_\theta$
            ChangeSlope(slope of $\overline{ac}$)
        case 5: $a$ hits $e$
            if an enpoint $p$ lies in $\angle vac$ then Add(p, f)
            e $\leftarrow$ first t-edge hit by $\overrightarrow{av}$
        case 6: $a$, $c$, and $c'$ become collinear
            For each t-edge that hits $\overline{cc'}$ in order from $c$ to $c'$
                        if an endpoint $p$ lies in $\angle vac'$ then Add($p$, b)
            change pivot $c \leftarrow c'$, $c' \leftarrow next(c')$
loop
repeat program, since path has not reached $\overline{u'v'}$
```

Table 3: Computing the minimum link path between inflection edges $\overline{uv}$ and $\overline{u'v'}$

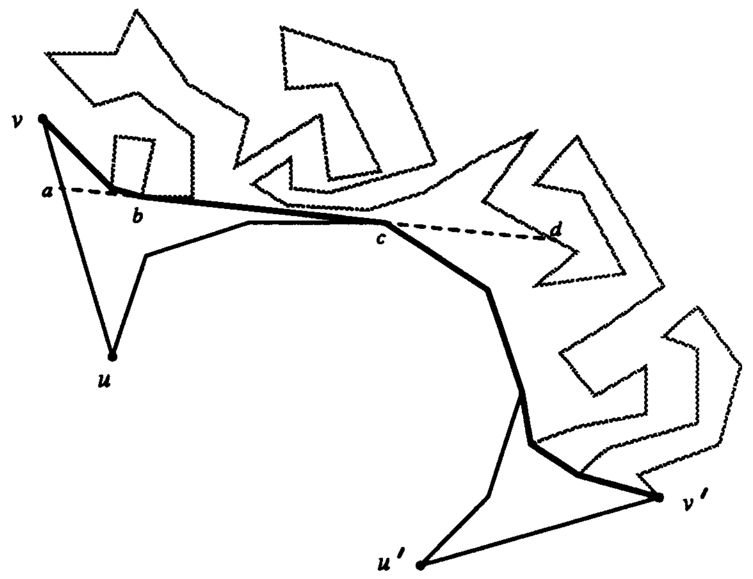


Figure 9: The shortest path γ has an inflection edge $\overline{bc}$

Lemma 4.3 *The point b first encountered by the sweeping tangent is the endpoint of a triangulation edge that crosses the segment $\overline{ua}$ or the chain from u to c.*

Proof: Because a triangulation has no reflex vertices, the tangent segment $\overline{ac}$ must cross a triangulation edge incident to b before it touches b. Since the segments $\overline{ua}$ and $\overline{ac}$ and the concave chain from u to c form a closed region, shown in figure 10, the lemma holds. ∎

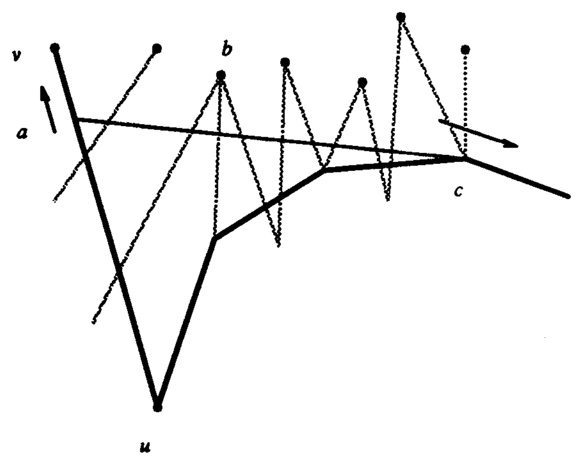


Figure 10: The sweep stops at b

Lemma 4.3 implies that we need look only at the convex hull of the endpoints of triangulation edges that we encounter during the sweep. These endpoints appear on the hull above $\overline{ac}$ in the same order as their edges appear along $\overline{ac}$. Points are added only at the ends of the segment $\overline{ac}$, so we can maintain the convex hull by a Graham scan [19] in a deque. Furthermore, the slope of $\overline{ac}$ changes monotonically, so we can also maintain b_θ, the point of the hull having a tangent with this slope. When $\overline{ac}$ hits b_θ, then $\overline{b_\theta c}$ is an inflection edge that lemma 4.2 says can be used in a minimum link path.

The above arguments establish the correctness of the algorithm outlined in table 3. To establish the running time, notice that the amount of work required to find a segment of the minimum link path is proportional to the number of triangulation edges that intersect the region depicted in figure 10. Since this region is free of points, these edges must intersect either $\overline{ua}$ or $\overline{ac}$. Since these segments are part of the minimum link path, we can charge this work to the number of triangles crossed by the computed path and obtain theorem 4.1.

Theorem 4.1 *A minimum link path α', homotopic to α, can be computed in $O(C_\alpha + \Delta_\alpha + \Delta_{\alpha'})$ time and space.*

In a simply connected BTM, a minimum link path can cross any triangulation edge at most three times: any path that crosses a triangulation edge e four or more times can be shortcut by a portion of e, decreasing its length without changing its homotopy class since all paths with the same starting and ending point have the same homotopy class. Thus, the total time to compute minimum link paths in simple polygons is linear. Among many obstacles, a minimum link k-gon can intersect $\Theta(kn)$ triangulation edges, as shown in figure 11.

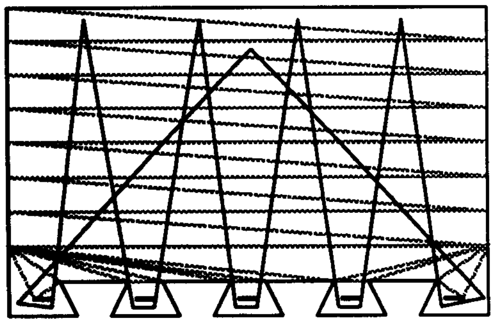


Figure 11: The minimum link k-gon intersects $\Theta(kn)$ edges

4.2 Minimum link closed curves

As in section 3.2, if we know a vertex or edge of the minimum link closed curve, we can use the path algorithm to compute it. When the minimum link curve is convex, however, it seems difficult to find such a vertex or edge.

Because of the algorithm of section 3.2, we can assume that our closed curve α is the minimum Euclidean curve of its homotopy class. If α has an inflection edge, then we can use the path algorithm of table 3 to find the paths between inflection edges. Lemma 4.1 implies that the resulting closed curve is a minimum link curve.

If α has no inflection edges, then the minimum link curve is convex. One can use the technique of Aggarwal et al. [1] as extended by Ghosh [17] to find the minimum link curve.

Briefly, one can find a curve by starting at some point p and finding the minimum link path back around to p by the algorithm of the previous section. The resulting path has at most one segment more than the minimum. Then one rotates the curve, keeping track of its points of contact with the inner and outer chains, to see if one can shorten the curve. The algorithm finds a minimum link closed curve with k line segments in $O(n \log k)$ time. It would be interesting to discover a matching lower bound.

5 Paths with restricted orientations

For some applications, such as VLSI, the paths are restricted to c fixed directions or orientations; we call such paths *c-oriented*. Rectilinear paths with the four orientations of north, south, east and west are the most common. In this section, we show that the universal cover is also a good tool for finding minimal c-oriented paths of a given homotopy class.

First, in section 5.1, we define convex polygonal distance functions appropriate to a given set of orientations. Then we show in section 5.2 that the length, under a convex distance function, of the path computed in section 3.1 equals the length of the shortest c-oriented path. Section 5.3 shows how to modify the minimum link algorithm of section 4.1 to compute minimum link c-oriented paths. Finally, section 5.4 shows that for paths restricted to three directions and for rectilinear paths, each homotopy class has a shortest path that is also a minimum link path. Mark de Berg [10] has independently noted this fact for rectilinear paths in simple polygons.

In all of the following sections, when we wish to construct paths restricted to c orientations explicitly, then we also restrict the boundary of the obstacles to the same set of c orientations. With such a restriction, there is always a path with at most $O(n)$ segments that follows obstacle boundaries. Without such a restriction, one can construct examples where any c-oriented path joining a given pair of points has infinitely many line segments.

5.1 Metrics versus distance functions

When paths are restricted, the link metric remains the number of line segment of a path. The Euclidean metric, however, should be replaced by a distance function that gives the length of the shortest restricted path between two points. One example is the Manhattan or L_1 metric for rectilinear paths, in which the length of a vector v is the sum of the lengths of the projections of v on the horizontal and vertical axes.

More generally, we can use Minkowski's convex distance functions [5]. Let A be a convex set whose interior contains the origin. The length of a vector v with respect to A is the amount that A must be scaled to include v; that is, $\|v\|_A = \liminf\{\lambda \geq 0 : v \in \lambda A\}$. The distance from point r to s is $\|s - r\|_A$. We do not call this a metric because it need not be symmetric: $\|v\|_A$ may not equal $\|-v\|_A$. It does, however, satisfy the triangle inequality [5]: if $u + v = w$ then $\|u\|_A + \|v\|_A \geq \|w\|_A$.

The points of the boundary of A are precisely the unit vectors of the distance function $\|\cdot\|_A$ [5]. Thus, choosing A to be the unit circle gives the Euclidean metric; the diamond defined by convex hull of the unit vectors in the axial directions gives the L_1 metric. For a c-oriented path, a path restricted to follow the orientations of c unit length *basis vectors*, we choose A to be the convex hull of the basis vectors. Since the convex hull is a c-gon that contains the origin in its interior, any vector v can be expressed as a linear combination of basis vectors with positive coefficients. The next lemma proves that the length $\|v\|_A$ is the minimum sum of coefficients over all positive linear combinations of basis vectors that equal v and that this length is realized by using the one or two basis vectors adjacent to v in circular order.

Lemma 5.1 *Let U be a circularly-ordered set of basis vectors defining a convex distance function. Let v be a vector in the cone defined by adjacent basis vectors u and u'. If $v = au + bu'$, then $\|v\|_U = a + b$.*

Proof: If v has the direction of a basis vector u, that is $v = au$ for some positive a, then $\|v\|_U = a$ by the triangle inequality.

Otherwise v lies in the cone defined by u and u'. Suppose that v is expressed by a positive linear combination involving a third basis vector u'', that is $v = au'' + w$, with the coefficient $a > 0$ and vector w equal the remaining terms of the positive linear combination. The path following vectors au'' and then w to v starts outside the cone and then enters by crossing, say, u. Therefore, $su = au'' + tw$ and $v = su + (1-t)w$. By the triangle inequality, however, $\|su\|_U + \|(1-t)w\|_U \leq \|au''\|_U + t\|w\|_U + (1-t)\|w\|_U$. In words, the length of the combination cannot increase by eliminating other basis vectors.

After eliminating all vectors but u and u', the vector v can be expressed uniquely. This proves the lemma. ∎

5.2 Shortest paths under a convex distance function

We use lemma 5.1 to find the length of the shortest path of a given homotopy type under a convex distance function. As before, we first compute the Euclidean shortest path α from p to q and use it as the representative of the homotopy class. This takes $O(C_\alpha + \Delta_\alpha)$ time.

The proofs leading to theorem 3.1 use only the triangle inequality to show that the path computed in section 3.1 is minimum under the Euclidean metric. But this implies

Theorem 5.1 *The Euclidean shortest path computed in section 3.1 is a shortest path under any convex distance function.*

To compute a c-oriented path of minimum length, it is enough to cut the Euclidean shortest path at all points with tangent vectors that are among the c basis vectors and compute c-oriented paths for the resulting pieces:

Lemma 5.2 *Let t be a point of the Euclidean shortest path α having a basis vector v as a tangent vector. Any minimum length path under the convex distance function goes through t.*

Proof: Slice the universal cover into three pieces by a line through t and parallel to v. Any path from p to q crosses the segment both before and after t. By applying lemma 5.1 to the cone containing v alone, the shortest path from the first to last crossing under the convex distance function is inside the segment. ∎

We can perform this cutting by simply traversing the Euclidean path—think of driving a car along the path, as in Guibas, Ramshaw, and Stolfi's kinetic framework for computational geometry [21], and cutting it whenever your direction is one of the c basis vectors. (If c is not constant, the time complexity is $O(C_\alpha \log c)$ because we must search for the slopes of edges.)

The slopes on each resulting path lie in a cone defined by two adjacent basis vectors. Thus, by lemma 5.1, each path should be replaced by a path using only those two orientations. If we are unconcerned about the number of line segments, then we can approach the Euclidean shortest path arbitrarily closely with those two orientations. More likely, however, one would want to use the methods of the next section to find a shortest path with few links; more on this in section 5.4.

5.3 Minimum link c-oriented paths

We use a greedy method quite similar to that of section 4.1 to construct c-oriented paths. The time required will be proportional to the number of triangles explored, which, in this case, may be more than the number of triangles intersected by the path. However, the worst-case bounds are similar to those of section 4.1: If c is a constant, $O(nk)$ time is sufficient to construct a k link path of a given homotopy class and $O(n + k)$ time is sufficient in a simple polygon. If c is not a constant, but the directions are initially sorted, the time bounds increase to $O(nk \log c)$ and $O(n \log c + k)$.

The next lemma begins to illustrate why more triangles must be explored. It shows that there are two directions in which a minimum link c-oriented path can start, namely those of the adjacent basis vectors whose cone contains the first edge of the Euclidean shortest path. We will have to try both directions and see which leads to the shorter path.

Lemma 5.3 *A minimum link path from p to q and homotopic to the Euclidean shortest path α can always begin with an edge whose orientation is the first edge clockwise (cw) or counterclockwise (ccw) of the first edge of α.*

Proof: Cut the universal cover from p in the directions of the edges adjacent to the first edge of α in cw and ccw order. Of the resulting two pieces (three if p was on the boundary of the universal cover), the one that contains q contains the Euclidean shortest path to q. Because the cuts are along adjacent basis vectors, the only way a c-oriented path from p can enter this piece is to cross one of the cuts. By using a part of the cut, one can shorten a path without adding extra links. Thus, we can assume that a minimum link path begins with an edge in one of the two cut orientations. ∎

We can try both directions at once in an inductive fashion. Suppose we know that a minimum link path from p to q can be made to pass through the point r in one of two adjacent orientations. Then we can construct a point r', one link farther along the minimum link path from p to q, with the same property: any minimum link path from p to q can be made to pass through r' in one of two adjacent orientations.

This situation is illustrated in figure 12. Any minimum link path can be made to pass through r with adjacent orientation vectors u and v. Draw line segments (or rays) in the universal covering space from r in these two directions and let r_u and r_v be the other endpoints. Cutting along segments $\overline{rr_u}$ and $\overline{rr_v}$ separates p from q, so the Euclidean shortest path α crosses one (and exactly one) of them at a point s. Without loss of generality, we assume that α turns right at a point t after s and continues to do so until it reaches an inflection edge e.

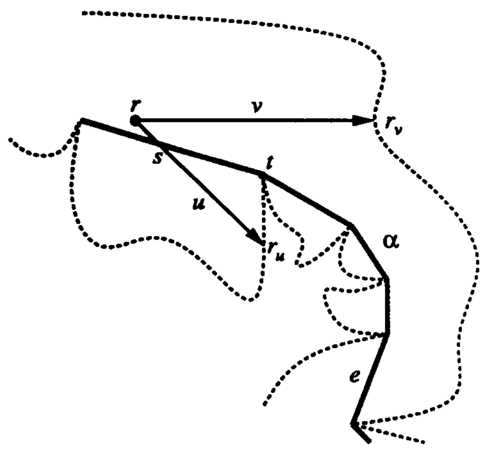


Figure 12: Minimum link paths go through r

Next we look at three canonical ways to extend the minimum link path from the point r. We prove lemma 5.4, which says that one of these three extensions can always be used by a minimum link path from p to q. The construction of another point r' that is one link farther along any minimum link path will come as a corollary of lemma 5.4.

Lemma 5.4 *Suppose a minimum link path α from p to q passes through r in one of two adjacent directions, that is, in either u or v. Then there is a minimum link path that leaves r in one of three canonical ways illustrated in figure 13 and described below.*

Proof: We describe the three extensions and show that, based on the direction that α turns after leaving r, one of the extensions can be used by a minimum link path.

1. One can sweep in direction u with a point a from r (or from s if s is on $\overline{rr_u}$) and maintain a segment from a to α in direction v as shown in figure 13(1). This sweep stops when a reaches r_u or the segment from a to α encounters a point b to the right of α. At best, b may equal t. Cutting the universal cover from b in direction v separates r from q, so any path to q must cross this cut. Since u and v are adjacent orientations, any path that follows direction u from r needs at least two turns to cross the cut—the minimum link path α might as well make them at a and at the cut.

2. One can sweep with the point a' from s to the left of α and maintain a tangent to α as in section 4.1. (This may mean first sweeping in direction $-u$ to r before sweeping in direction v.) Consider the set of tangents defined by the moving point a' and a vertex where α turns right between s and e. If this set contains any tangents parallel to a basis vector, then let w be the last basis vector, b be the point of tangency, and a be the point on $\overline{rr_v}$, as illustrated in figure 13(2). Again, cutting the universal cover along $ext(\overline{ab})$ separates r from q. Any path that follows v and then a direction from u clockwise to w takes at least two turns to cross the extension—the minimum link path α might as well use the extension.

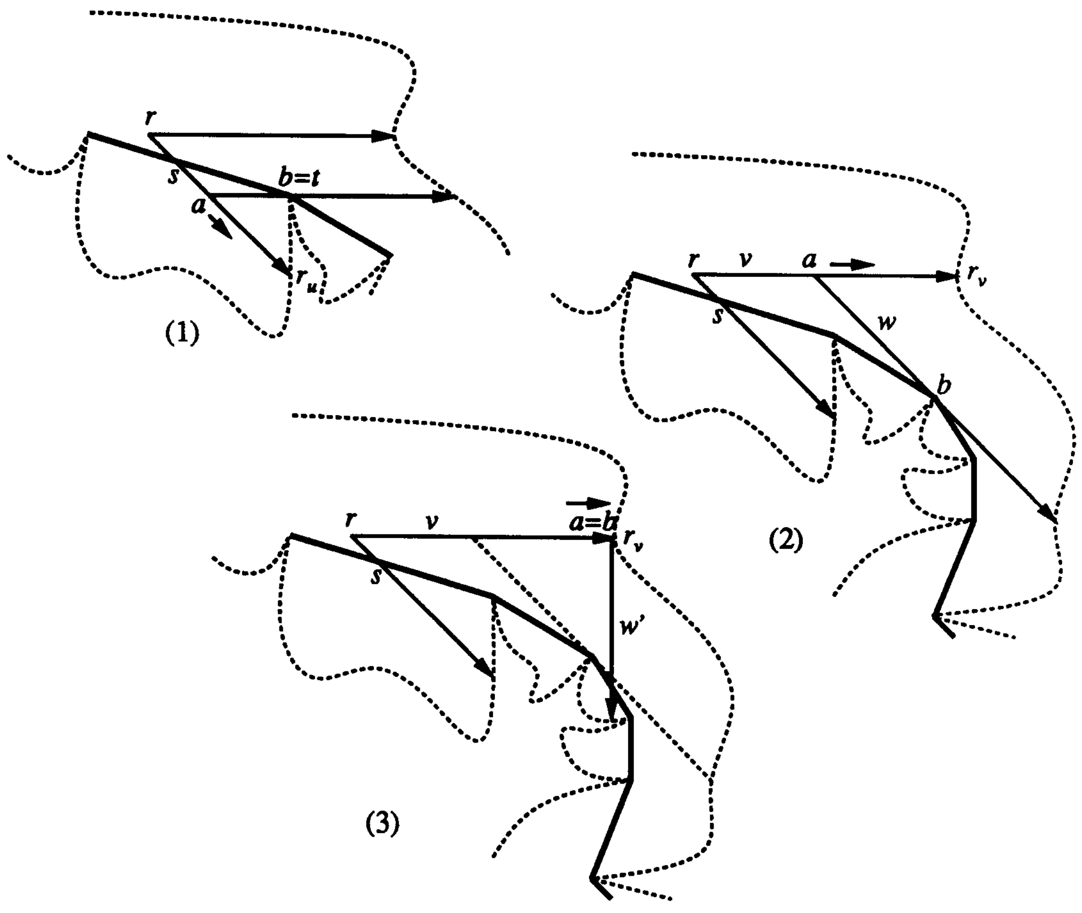


Figure 13: Three canonical ways to extend the minimum link path

3. Let w' be the direction clockwise of and adjacent to the last basis vector tangent w, if it exists, or $w' = u$ if it does not. Then, as in the first case, sweep in direction v with a point a from r and maintain a segment from a to α in direction w' as shown in figure 13(3). This sweep stops when a reaches r_v or the segment from a to α encounters a point b to the left of α. Because there is no tangent in direction w', cutting the universal cover from b in direction w' separates r from q. Any path that follows v from r and then turns at or clockwise of direction w' needs at least two turns to cross the cut—the minimum link path α might as well make them at a and at the cut.

■

The algorithm presented in table 3 can be used almost directly to find the last basis vector that is tangent to the Euclidean shortest path. To modify the algorithm to perform the sweep with a line of fixed slope, as desired in methods (1) and (3), is tedious but straightforward.

We can observe that, if method (2) applies, then we need not use method (1). The following corollary uses this fact to find another point that a minimum link path can be made to go through.

Corollary 5.2 *Suppose a minimum link path α from p to q passes through r in one of the adjacent directions u or v. Then the minimum link path can be made to pass through a point r', one turn farther along the path, in one of two adjacent directions.*

> **Proof:** When a basis vector is tangent to the path α in lemma 5.4, method (2), then the point of tangency is at or beyond t and the slope of the tangent is clockwise from v. Thus, any path from p to q will cross before the extension $ext(\overline{ab})$ after the segment from t in direction v, which is the best that the first method can do. Therefore, a greedy approach will take the second rather than the first option whenever possible.
>
> Thus, if there is no basis vector tangent, the point r' is the intersection of the paths formed by the first and third methods of lemma 5.4. Otherwise, r' is the intersection of the paths of the second and third methods. ∎

From these pieces, it is not difficult to form the overall algorithm. Lemma 5.3 says that initially we start at the point p and have two choices for the initial direction. Whenever we are in such a situation, we look for a basis vector tangent and apply methods (2) and (3) or methods (1) and(3) of lemma 5.4. By corollary 5.2 we reach another point in which we have two choices.

If we reach the destination q in $k-1$ repetitions, then the minimal link c-oriented path has k line segments. The number of triangles explored is at most linear at each step, so $O(nk)$ time is sufficient, if c is a constant. If c is not constant, then we must account for the work to find the last basis vector tangent—the time bound increases to $O(nk \log c)$, assuming the c orientations are given in a form that permits binary search.

Two remarks close this subsection.

First, the analysis of the running time can be sharpened in a simple polygon if there are two opposite basis vectors. If we use the algorithm of Fournier and Montuno [15] to change the triangulation to a trapezoidation with sides parallel to these basis vectors, then c-oriented paths can follow the edges of the trapezoids. Now, application of lemma 5.4 computes two minimum link paths from r to r'. If we travel from p to q and arbitrarily choose which path to take at each intermediate point r, we follow a path that has at most $2k$ links—it is at most double the optimal path. In fact, the subpath between any two intermediate points is at most double the optimal subpath. Any trapezoid edge that intersected more than six edges of the path could be used to shorten the subpath—thus no trapezoid is explored more than a constant number of times. The running time of the algorithm in this case is $O(n+k) = O(n)$ when c is constant and $O(n \log c)$ when c is not.

Second, when c is a constant, it is theoretically possible to compute the path in time proportional to the time needed to compute the Euclidean shortest path. ("Theoretically possible" meaning that one would never want to perform the computation this way.) Compute c different trapezoidations with their sides parallel to the c basis vectors and find the Euclidean shortest path in all c trapezoidations. When applying lemma 5.4 to compute the c-oriented path, use the trapezoidation in direction u in performing the computation of method (1) and use that in direction v in the computation of methods

(2) and (3). Every trapezoid inspected during the computation intersects the Euclidean shortest path in that trapezoidation, so the total time is proportional to computing the Euclidean paths.

5.4 Simultaneous minimization of length and links

We have seen how to compute the length of the shortest c-oriented path under a convex distance function and how to compute minimum link c-oriented paths. In this section we remark that a simplified version of the minimum link path algorithm can compute the path with fewest links of all shortest c-oriented paths. For rectilinear paths and paths restricted to three directions, we prove that this path is also a minimum link path—that length and links are minimized simultaneously.

Section 5.2 showed that the shortest c-oriented path under a convex distance function can be obtained by breaking the Euclidean shortest path at all vertices with basis vector tangents and approximating each piece by a path that follows two adjacent orientations. We can use a simple version of the algorithm of the previous section to compute such paths—only methods (1) and (3) of lemma 5.4 ever apply. By combining the resulting paths appropriately, merging collinear segments into single links, one can compute the shortest c-oriented path with the fewest links.

To determine when this path is also a minimum link path, we make the following definition. A member u of a set of basis vectors U satisfies the *halfplane condition* if there is a halfplane that contains all of U except u. Now, consider driving along the Euclidean shortest path, moving and turning in accordance to Guibas, Ramshaw, and Stolfi's kinetic framework for computational geometry [21], and noting which in which basis vector orientations you face during a turn about a vertex. If the last basis vector at a vertex has the halfplane condition, then both the minimum length and minimum link paths can pass through that vertex in the direction of the basis vector.

Lemma 5.5 *If the last basis vector, v, of a point t on the Euclidean shortest path α satisfies the halfplane condition, then a minimum link c-oriented path homotopic to α has an edge passing through t in direction v.*

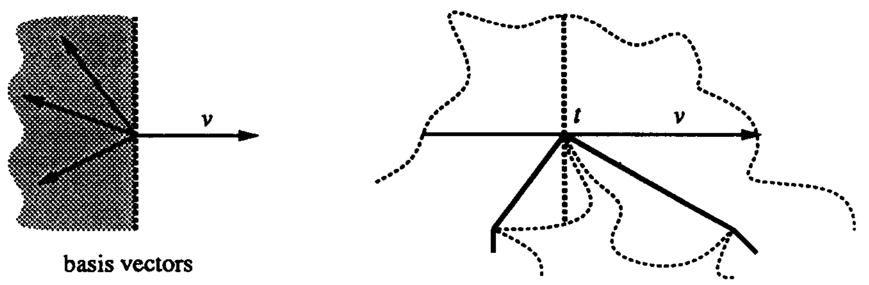


Figure 14: The halfplane condition

Proof: Cut the universal cover through t along a line that defines a halfplane containing all the basis vectors except v, as illustrated in figure 14. If any c-oriented path crosses this cut at t, then it must do so in direction v by the halfplane condition and the fact that v is the last basis vector.

If the crossing is not at t, then we can move it to t without increasing the number of links as follows. Because of the halfplane condition, a c-oriented path must cross the cut using an edge e that is parallel to v. Separate the universal cover into three pieces by a line segment through t and parallel to v. We can shorten the path by a portion of this line segment; the number of links does not increase because edge e is cut off the path. ■

For rectilinear paths and any convex distance function defined by three vectors, all vectors satisfy the halfplane condition and lemma 5.5 implies that a minimum link path goes through all the points with tangents that are basis vectors. To compute the minimum link c-oriented path, we could cut the Euclidean shortest path at all these points and compute the path greedily. This, however, is precisely the computation of the shortest path with fewest links—the path computed is minimum with respect to the convex distance function and the link metric simultaneously.

6 Conclusions

We have shown that the universal covering space of a triangulated region gives a useful framework for optimizing paths in the region under the Euclidean and link metrics. We have given simple direct algorithms for Euclidean shortest path trees and minimum link paths that use arrays in place of finger search trees. We have also given the first algorithms for computing minimum length and link c-oriented paths.

Acknowledgements

We would like to thank Joseph Mitchell and Leonidas Guibas for discussions on these and related problems. Both authors thank Digital Equipment Corporation and ALCOM for supporting this research.

References

[1] Alok Aggarwal, Heather Booth, Joseph O'Rourke, Subhash Suri, and Chee K. Yap. Finding minimal convex nested polygons. *Information and Computation*, 83(1):98–110, October 1989.

[2] M. A. Armstrong. *Basic Topology*. McGraw-Hill, London, 1979.

[3] B. Baumgart. A polyhedral representation for computer vision. In *Proceedings of the AFIPS National Computer Conference*, pages 589–596, 1975.

[4] Binay Bhattacharya and Godfried T. Toussaint. A linear algorithm for determinimg translation separability of two simple polygons. Technical Report SOCS–86.1, School of Computer Science, McGill University, Montreal, 1986.

[5] J. W. S. Cassels. *An Introduction to the Geometry of Numbers*. Springer-Verlag, Berlin, 1959.

[6] Bernard Chazelle. A theorem on polygon cutting with applications. In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*, pages 339–349, 1982.

[7] Bernard Chazelle. Triangulating a simple polygon in linear time. Technical Report CS–TR–264–90, Princeton University, Department of Computer Science, May 1990.

[8] K. L. Clarkson, S. Kapoor, and P. M. Vaidya. Rectilinear shortest paths through polygonal obstacles in $O(n \log^2 n)$ time. In *Proceedings of the Third Annual ACM Symposium on Computational Geometry*, pages 251–257, 1987.

[9] Richard Cole and Alan Siegel. River routing every which way, but loose. In *Proceedings of the 25th IEEE Symposium on Foundations of Computer Science*, pages 65–73, 1984.

[10] Mark de Berg. On rectilinear link distance. Technical Report RUU–CS–89–13, Rijksuniversiteit Utrecht, Box 80.089, 3508 TB Utrecht, May 1989.

[11] Mark de Berg. Translating polygons with applications to hidden surface removal. In *SWAT 90: 2nd Scandinavian Workshop on Algorithm Theory*, number 447 in Lecture Notes in Computer Science, pages 60–70. Springer-Verlag, 1990.

[12] Mark de Berg, Mark van Kreveld, Mark Overmars, and Bengt Nilson. Finding shortest paths in the presence of orthogonal obstacles using a combined l_1 and link metric. In *SWAT 90: 2nd Scandinavian Workshop on Algorithm Theory*, number 447 in Lecture Notes in Computer Science, pages 211–224. Springer-Verlag, 1990.

[13] P. J. de Rezende, D. T. Lee, and Y. F. Wu. Rectilinear shortest paths with rectangular barriers. *Discrete and Computational Geometry*, 4:41–53, 1989.

[14] E. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[15] Alain Fournier and Delphin Y. Montuno. Triangulating simple polygons and equivalent problems. *ACM Transactions on Graphics*, 3(2):153–174, 1984.

[16] Shaodi Gao, Mark Jerrum, Michael Kaufmann, Kurt Mehlhorn, Wolfgang Rülling, and Christoph Storb. On continuous homotopic one layer routing. In *Proceedings of the Third Annual ACM Symposium on Computational Geometry*, pages 392–402, 1987.

[17] Subir Kumar Ghosh. Computing the visibility polygon from a convex set and related problems. *Journal of Algorithms*, 1990. To appear.

[18] Subir Kumar Ghosh and David M. Mount. An output sensitive algorithm for computing visibility graphs. In *Proceedings of the 28th IEEE Symposium on Foundations of Computer Science*, pages 11–19, 1987.

[19] R. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1:132–133, 1972.

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

[21] L. Guibas, L. Ramshaw, and J. Stolfi. A kinetic framework for computational geometry. In *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science*, pages 100–111, 1983.

[22] Leonidas Guibas, John Hershberger, Joseph Mitchell, and Jack Scott Snoeyink. Using minimum link paths to approximate polygons. In preparation, 1990.

[23] Leonidas Guibas and Jorge Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, 1985.

[24] Leonidas J. Guibas and John Hershberger. Optimal shortest path queries in a simple polygon. *Journal of Computer and System Sciences*, 39(2):126–152, October 1989.

[25] Ralf Hartmut Güting. *Conquering Contours: Efficient Algorithms for Computational Geometry.* PhD thesis, Dortmund, 1983.

[26] S. Kapoor and S. N. Maheshwari. Efficient algorithms for Euclidean shortest path and visibility problems with polygonal obstacles. In *Proceedings of the Fourth Annual ACM Symposium on Computational Geometry*, pages 172–182, 1988.

[27] R. C. Larson and V. O. Li. Finding minimum rectilinear paths in the presence of barriers. *Networks*, 11:285–304, 1981.

[28] D. T. Lee and F. P. Preparata. Euclidean shortest paths in the presence of rectilinear barriers. *Networks*, 14(3):393–410, 1984.

[29] Charles E. Leiserson and F. Miller Maley. Algorithms for routing and testing routability of planar VLSI layouts. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, pages 69–78, 1985.

[30] Charles E. Leiserson and Ron Y. Pinter. Optimal placement for river routing. *SIAM Journal on Computing*, 12(3):447–462, 1983.

[31] Joseph S. B. Mitchell, Günter Rote, and Gerhard Woeginger. Minimum-link paths among obstacles in the plane. In *Proceedings of the Sixth Annual ACM Symposium on Computational Geometry*, pages 63–72, 1990.

[32] David M. Mount. The number of shortest paths on the surface of a polyhedron. *SIAM Journal on Computing*, 19(4):593–611, August 1990.

[33] James R. Munkres. *Topology: A First Course.* Prentice-Hall, Englewood Cliffs, N.J., 1975.

[34] Ron Y. Pinter. River routing: Methodology and analysis. In Randal Bryant, editor, *The Third Caltech Conference on Very Large Scale Integration.* Computer Science Press, Rockville, Maryland, 1983.

[35] Franco P. Preparata and Michael I. Shamos. *Computational Geometry—An Introduction.* Springer-Verlag, New York, 1985.

[36] G. Rawlins and D. Wood. Optimal computation of finitely oriented convex hulls. *Information and Computation*, 72:150–166, 1987.

[37] D. Richards. Complexity of single-layer routing. *IEEE Transactions on Computers*, C-33(3):286–288, March 1984.

[38] Jörg-Rüdiger Sack. *Rectilinear Computational Geometry.* PhD thesis, Carleton University, 1984.

[39] Micha Sharir and Amir Schorr. On shortest paths in polyhedral spaces. *SIAM Journal on Computing*, pages 193–215, 1986.

[40] S. Suri and J. O'Rourke. Finding minimal nested polygons. Technical report, The Johns Hopkins University, 1984.

[41] Subhash Suri. A linear time algorithm for minimum link paths inside a simple polygon. *Computer Vision, Graphics, and Image Processing*, 35:99–110, 1986.

[42] G. Toussaint. On separating two simple polygons by a single translation. *Discrete and Computational Geometry*, 4:265–278, 1989.

[43] Godfried T. Toussaint. Movable separability of sets. In Godfried T. Toussaint, editor, *Computational Geometry*, volume 2 of *Machine Intelligence and Pattern Recognition*, pages 335–376. North Holland, Amsterdam, 1985.

[44] Godfried T. Toussaint. Computing geodesic properties inside a simple polygon. *Revue D'Intelligence Artificielle*, 3(2):9–42, 1989. Also available as technical report SOCS 88.20. School of Computer Science. McGill University.

[45] Cao An Wang and Edward P. F. Chan. Finding the minimum visible vertex distance between two nonintersecting simple polygons. In *Proceedings of the Second Annual ACM Symposium on Computational Geometry*, pages 34–42, 1986.