Computing minimum length paths of a given homotopy class

John Hershberger
DEC Systems Research Center

Jack Snoeyink*
Department of Computer Science
University of British Columbia

Abstract

In this paper, we show that the universal covering space of a surface can be used to unify previous results on computing paths in a simple polygon. We optimize a given path among obstacles in the plane under the Euclidean and link metrics and under polygonal convex distance functions. Besides revealing connections between the minimum paths under these three distance functions, the framework provided by the universal cover leads to simplified linear-time algorithms for shortest path trees, for minimum-link paths in simple polygons, and for paths restricted to c given orientations.

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 shortest 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 paths, and convex and link distance functions for paths that are restricted to use c given orientations, such as rectilinear paths. Our work presents these distance functions in a unifying framework, a triangulation of the unversal covering space. In this framework, we can generalize results for simple polygons to compute shortest paths of a given homotopy class. We also simplify proofs and replace complicated data structures such as finger search trees by simple arrays and stacks.

We organize the paper around four variants on shortest path problems: Euclidean shortest paths and shortest path trees in section 3, minimum-link paths in section 4, shortest closed loops in section 5, and paths with restricted orientations in section 6. In the remainder of this section we review previous results on these problems and summarize our results using the universal cover. Section 2 gives formal definitions of the universal covering space and of triangulated manifolds, homotopy classes, and distance metrics—the important topological tools for our algorithms.

1.1 Euclidean shortest paths

Many researchers have investigated the problems of finding Euclidean shortest paths in simple polygons. Chazelle [7] and Lee and Preparata [33] gave a *funnel* algorithm that, in a triangulated polygon, computes the shortest Euclidean path between two points in linear time.

The funnel algorithm has been extended to handle one of the tractable cases of river routing in VLSI. Cole and Siegel [10], Leiserson and Maley [34], and Gao et al. [18] 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 [35, 42, 45]. Leiserson and Maley and Gao et al. use the funnel algorithm to compute the *rubber-band equivalent* of each wire as a basic preprocessing step.

In section 3.1 we describe the application of the funnel algorithm in the universal cover of a triangulated manifold. Then, in section 3.2, we extend it to efficiently maintain the shortest path homotopic to a

*Portions of the second author's research were performed at Stanford and Utrecht Universities.

path that is given on-line. Both of these algorithms take time proportional to the time needed to trace the representative of the path through the triangulation and both use simple data structures—arrays and stacks.

Guibas et al. [23] used finger-search trees 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. Our on-line shortest path algorithm can compute this shortest path tree using simpler data structures.

Finding minimum paths among obstacles when the homotopy class is not given is a more difficult problem, and is one that we will not discuss. For the Euclidean metric, one typically builds the visibility graph and searches it with Dijkstra's algorithm [16]; see Ghosh and Mount [21] and Kapoor and Maheshwari [31] for efficient algorithms.

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 [48] developed a linear time algorithm for computing the minimum path between two points in a simple polygon. Ghosh [19] 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. [23]. We show how to extend our Euclidean minimum path algorithm to compute the minimum-link path in time proportional to the number of triangles that this 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 [37] 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. Other recent work has considered combining link and Euclidean metrics [2, 36].

1.3 Shortest loops

There are special closed loops of interest to computational geometers that fit within the framework of this research. Under the Euclidean metric, the shortest loop enclosing a set of points or line segments is the convex hull of the set. The shortest loop enclosing a set and contained in the interior of a polygon is the *relative convex hull* of the set. Toussaint and others have studied relative convex hulls, also called geodesic hulls, in connection with the separability of polygons under translation [5, 12, 51, 52, 53]. Czyzowicz et al. [11] have solved the "Aquarium Keeper's Problem," a generalization of the problem of computing the minimum perimeter polygon that touches each edge of a given convex polygon. Essentially, they use the reflection principle to convert this problem to one of computing the shortest loop around a triangulated annulus or Möbius strip. Our results on closed loops simplify these solutions and generalize them slightly.

Minimum-link loops 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 Chan [54, 55] 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 [19] 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 [49] compute a minimum-link polygon separating an m-gon and its enclosing n-gon in $O(mn)$ time; we note (as did Ghosh and Maheshwari [20]) that this is actually the easy case and can be solved in linear time.

1.4 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. [46]) and in the presence of obstacles [9, 15, 32]. Mark de Berg [13] 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 [14] give a quadratic algorithm for a combined link and L_1 metric for paths among obstacles. 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 [15] and $O(n \log^2 n)$ if they are not [9].

Güting [29] defined c-oriented polygons as a generalization of rectangles; he and others [30, 44, 50] have looked at various geometry problems with restricted orientations. The recent survey of Nilsson et al. [40] summarizes many results.

We show that the universal cover is also a useful tool to compute shortest and minimum-link c-oriented paths of a given homotopy type. Specifically, we show (in section 6.2) that the shortest Euclidean path, measured under a convex distance function, has the length of the shortest c-oriented path. In section 6.3, we give an algorithm to compute minimum-link c-oriented paths and also show how to use it to compute a shortest c-oriented path from the shortest Euclidean path. We also look at conditions when a c-oriented path is simultaneously shortest and minimum-link (section 6.4).

1.5 Improved data structures for simple polygons

The algorithms that we develop for the universal cover have implications in the special case of triangulated simple polygons.

Section 3.2 By developing a dynamic version of the funnel algorithm, we obtain a linear time algorithm for shortest path trees that uses only a fixed size deque (doubly-ended queue) and a stack for storage.

Section 4 For minimum-link paths, where distance is measured by the number of line segments, we develop an output-sensitive algorithm that runs in linear time in a simple polygon and uses deques and stacks rather than visibility maps and shortest path trees.

Section 5.1 By walking around a loop two or four times, we compute the Euclidean shortest loop in both orientable and non-orientable manifolds without using shortest path trees.

Section 6 We compute minimum length and/or link paths restricted to c orientations in $O(n \log c)$ time.

2 Preliminaries

We begin by defining some important topological objects: triangulated manifolds, homotopy classes, metrics, and covering spaces.

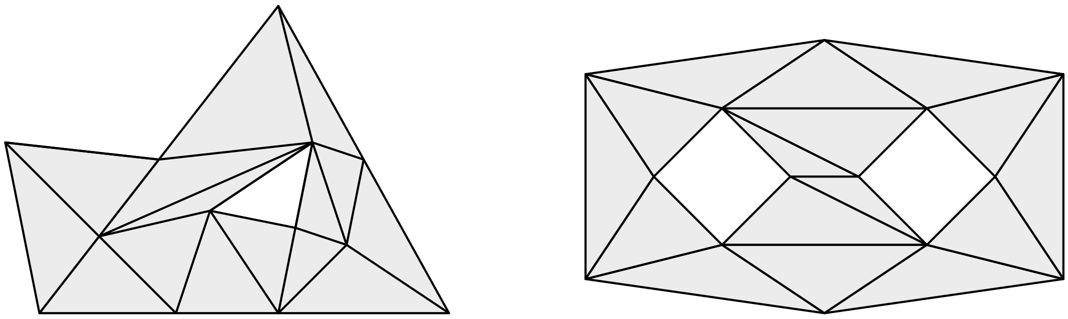
2.1 Manifolds and simplicial complexes

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.

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.*

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 boundary edges are adjacent to their triangles in the ordering. 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. Because vertices are the



a. b.

Figure 1: Two triangulated manifolds; the one on the right is a boundary-triangulated 2-manifold (BTM)

only source of curvature in a piecewise-linear surface, this implies that a BTM is *flat*—the neighborhood of any point looks like a portion of the Euclidean plane [28, 41].

One can represent a BTM, or any other 2-d simplicial complex, in a computer using Guibas and Stolfi's quad-edge structure [25], Baumgart's winged-edge structure [4], or the dual graph of the simplicial complex. In our algorithms, we require that each triangle of M be able to access its incident edges and each edge of M its incident triangles in constant time. If a polygonal region R is given, we can triangulate R and construct one of the above representations of the triangulation in $O(n \log n)$ time by a sweepline algorithm [43] or, if R has a constant number of boundary components, in linear time by Chazelle's algorithm [8].

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. Informally, 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—relating paths that are *homotopic.*

2.2 Homotopy classes

The topological concept of *homotopy* formally captures the notion of deforming paths. 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 [3, 39].

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 and most importantly, 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, we define a *closed loop* to be 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 loops 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 $\Gamma(x,1) = \beta(x)$. We use this definition only in section 5. (A *closed loop* is different from a path with the starting and ending points identitfied, because our definition of path homotopy never moves the endpoints of a path.)

One could go on to define homotopy in M for two subdivisions α and β—indeed, we do so in a paper with Guibas and Mitchell [27] and show that computing minimum-link subdivisions is NP-hard.

A homotopy relation partitions paths or closed loops into equivalence classes. Thus, we can describe a homotopy class by giving a representative path or loop 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 triangulation edges; we can form a *canonical path* that visits the midpoints of triangulation 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.

We 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 ([3, 39]) *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 points are homotopic.*

2.3 Path complexity and metrics

In computer applications, paths are most often specified as a sequence of line segments or pieces of low-degree polynomials. We define the complexity of a path α, denoted C_α, to be the number of pieces that compose α. For a path α in a BTM M we also count Δ_α, the number of times that α crosses a triangulation edge of M. For the canonical path defined above, we have $C_\alpha = \Delta_\alpha$, but, in general, either one of the two quantities could be greater.

We assume that α is represented in the computer in some form that can be traced through the BTM data structure in time proportional to $O(C_\alpha + \Delta_\alpha)$. For example, if α is piecewise linear, then for each segment in each triangle we can compute a constant number of segment/segment intersection points to determine whether we need to advance to the next segment or to the next triangle. Storing the vertices of α in an array and the BTM M in any of the data structures mentioned above permits tracing 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 path or loop is the sum of the lengths of its pieces in all the triangles it intersects. In the *link* metric, the length of a path or loop is the number of its line segments. Because BTMs are flat, the minimum length paths under both metrics 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 [38, 47].

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 6.

2.4 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 $p\colon U \to X$ 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.

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:

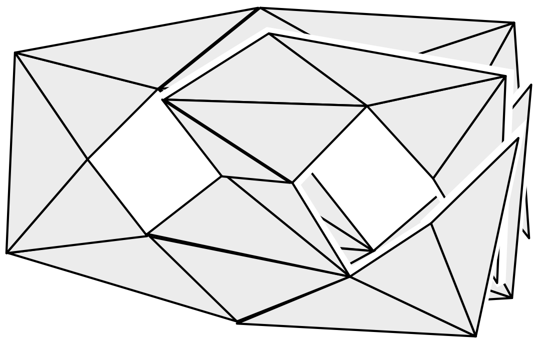


Figure 2: A portion of the universal cover

Lemma 2.3 *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, subpaths of α start and end at the same edge; we can deform these subpaths 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. ∎

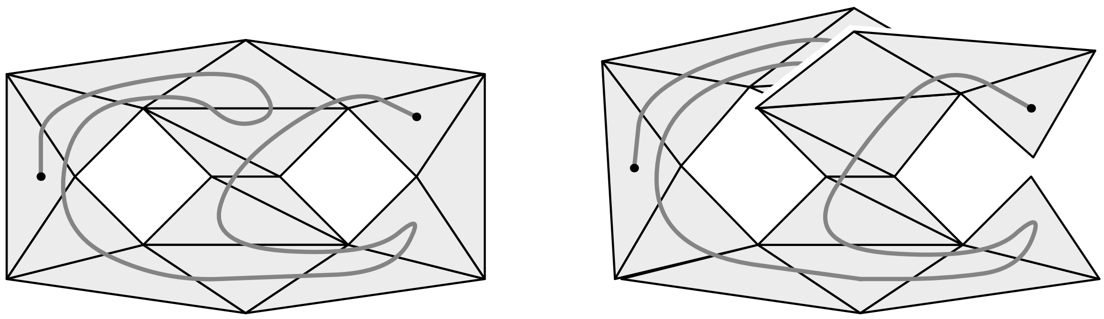


Figure 3: The lift of α

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 2.4 *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, trace 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 begin by applying these topological tools to the *funnel* algorithm, developed by Lee and Preparata [33] and Chazelle [7] and used by many researchers to find shortest paths [18, 23, 26, 34]. Section 3.1 reviews this algorithm and remarks that it can be used to find shortest paths between two points of a given homotopy type. Section 3.2 extends this algorithm to maintain the shortest path homotopic to a path that is given on-line. As a by-product, we can find shortest path trees in linear time without using finger search trees. This simplifies an important algorithm of Guibas et al. [23].

3.1 Funnels and the shortest path between two points

First we review *funnels,* defined by Lee and Preparata [33]. 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. We store the vertices of a funnel in a double-ended queue, a *deque.*

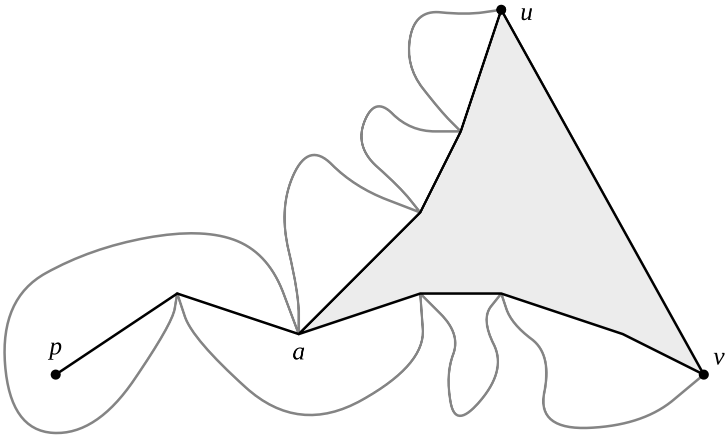


Figure 4: A funnel

Figure 5 shows that the extensions of funnel edges define wedges. If we cross the segment $\overline{uv}$ into a triangle $\triangle uvw$, then we would like to obtain the shortest path to w to construct the funnel for the segment $\overline{uw}$ or $\overline{vw}$. To find the funnel for $\overline{uw}$, we pop points from the v end of the deque until we reach b, the apex of the wedge that contains w, then we push w. If the apex of the previous funnel is popped during the process, then b becomes the new funnel apex. Notice that the edge $\overline{bw}$ is on the shortest path from p to w.

The shortest path algorithms of Chazelle [7] and Lee and Preparata [33] both look for a path in a *sleeve* polygon—a triangulated simple polygon whose dual tree is a simple path. We shall look for a path in a *sleeve* BTM.

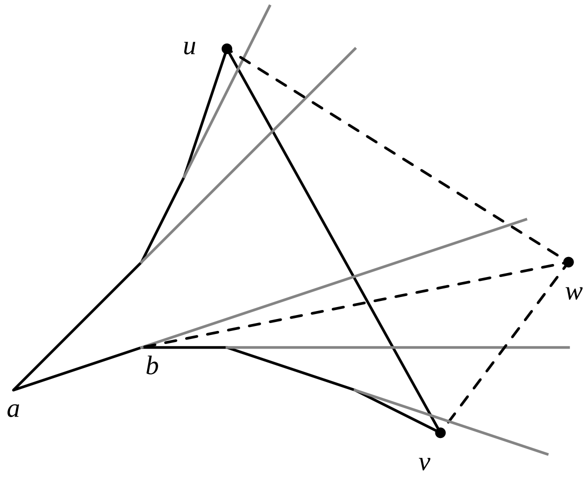


Figure 5: Splitting a funnel about w

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 2.4.

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, the lift of 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, under projection, 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 3.2 *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 On-line shortest paths and shortest path trees

In this section, we show how to maintain the deque that represents the funnel for a path α that is given on-line. We wish to trace α through the universal cover in $O(C_\alpha + \Delta_\alpha)$ time, as above. Since, however, we do not know the entire path ahead of time, we must be able to handle doubling back over the same triangle edge many times; we cannot afford to do more than a constant amount of work to update the deque each time.

Besides being useful for interactive applications, this procedure can be adapted 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. [23] compute the shortest path tree of any triangulated simple polygon by splitting funnels—they use finger search trees to find the splitting vertices and split the funnels 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 uses arrays in place of finger search trees and still runs in linear time. We describe first the data structure and then the algorithm that uses it.

We use an array and a history stack to support five operations on a deque that stores a funnel.

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

We store the deque in the entries of an array with indices from *first* through *last*. When we perform an **Add**() or **Split**(), we record the previous values of changed array entries and/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 deque, denoted by indices $\mathit{first} = n$ and $\mathit{last} = n - 1$, and perform at most n **Add**() operations, then an array of size $2n$ is sufficient to hold the deque.

Length(*deque*)
  return $\mathit{last} - \mathit{first} + 1$

Add(f, *deque*, x)
  decrement *first*
  push (**Add**, f, *deque*[*first*]) to stack
  set *deque*[*first*] $\leftarrow x$

Split(f, *deque*, i)
  check $0 \le i <$ **Length**(*deque*)
  push (**Split**, f, *last*) to stack
  set $\mathit{last} \leftarrow i + \mathit{first}$

Index(*deque*, i)
  check $0 \le i <$ **Length**(*deque*)
  return *deque*[i + *first*]

Undo(*deque*)
  if stack top is (**Add**, f, x)
    set *deque*[*first*] $\leftarrow x$
    increment *first*
  else stack top is (**Split**, f, i)
    set $\mathit{last} \leftarrow i$

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

Suppose the path α begins at point p in a BTM M. The algorithm will trace α through the universal cover according to lemma 2.4—beginning with the *base* triangle that contains p. Notice that whenever the lift of α is in the base triangle, the funnel deque should consist only of p. Whenever the lift of α crosses an edge $\overline{uv}$ out of the base triangle, we add the endpoints to the funnel deque by **Add**(f, *deque*, u) and **Add**(b, *deque*, v).

Suppose the path α crosses an edge $\overline{uv}$ into a new triangle $\triangle uvw$ that is added to U_α. If α later leaves through one of the other edges of $\triangle uvw$, then the current funnel is split into the funnels defined by $\overline{uw}$ and by $\overline{vw}$ as illustrated in figure 5. The key observation is that whenever the lift of α is in $\triangle uvw$, the index i of where the deque is to split is the same. We use an increasing increment search to compute this index i: 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 $d = \min\{i, \textbf{Length}(\mathit{deque}) - i\}$. Finger search trees were used in [23] to implement the simultaneous increasing-increment search, but arrays avoid the extra pointer complexity. We store this splitting index with $\triangle uvw$ in U_α.

Now, 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 two operations: usually a split/add pair, but an add/add when α is returning to the base triangle. Lemma 3.3 establishes that this on-line algorithm and the previous section's off-line algorithm compute the same funnel.

Lemma 3.3 *For a curve α from p to q in a BTM M, the on-line algorithm computes the funnel corresponding to the sleeve of the path from p to q in the universal cover of M.*

Proof: We prove this lemma by induction on the number of triangulation edges that α crosses. The induction hypothesis is that the pairs of operations that have been placed on the history stack are exactly those that would be performed by the off-line algorithm. This is trivially true if α is entirely contained in the base triangle.

Suppose the invariant holds for all paths crossing k triangulation edges, and let α be the concatenation of α', which crosses k triangulation edges, and α'', which crosses one edge. If α'' traverses an edge away from the base triangle, then the sleeve of α is the sleeve of α' with one new triangle added to the end. The split/add (or add/add) performed and put on the history stack establishes the hypothesis for α. Otherwise, α'' traverses an edge towards the base triangle. Since the sleeve of α is the sleeve of α' minus the last triangle, undoing the last two operations performed and removing them from the history stack does the right thing. ■

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 analysis of Guibas et al. [23] can be applied here to show that the time to find the splitting indices is linear in Δ_α. In brief, the time to compute splitting indices for the triangles of U_α is bounded by $T(\Delta_\alpha)$ where

$$T(n) \le \max_i \{T(i) + T(n-i) + \log\min\{i, n-i\}\}.$$

Thus, we have established the following theorem.

Theorem 3.4 *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 each of the vertices of the polygon—that is, the shortest path tree of P.

4 The link metric

In this section we show how to compute the minimum-link path, α', homotopic to a given path α in time proportional to $C_\alpha + \Delta_\alpha + \Delta_{\alpha'}$, the complexity of the path α plus the number of triangles intersected by both paths. Our approach is inspired by Ghosh's [19] 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, $O(C_\alpha + \Delta_\alpha)$, 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. (The shortest path has complexity at most $\Delta_{\alpha'}$, so we perform all remaining complexity analysis in terms of $\Delta_{\alpha'}$.) 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 the universal cover:

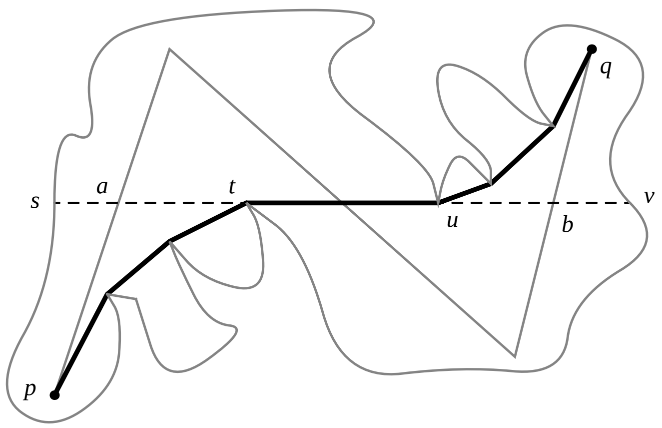


Figure 6: Extension $ext(\overline{tu})$ separates U

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 6. 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. ■

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 7.

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.

Lemma 4.2 *The minimum-link path joining $\overline{uv}$ and $\overline{u'v'}$ either has zero or one segments or it 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 7. (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 $\overline{cd}$. If we shortcut the path by following the subsegment of $\overline{ad}$ from a, through c, to the intersection of the path with $\overline{cd}$, then we do not increase the number of line segments on the path. ■

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 2 and described in the rest of this section.

Notice that $\overline{ac}$ is tangent to the concave chain. We find $\overline{ac}$ 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:

Variables:
Points a, c, c':
a sweeps "upward" along segment $\overline{uv}$.
$\overline{ac}$ is tangent to the concave chain at c.
Point c' follows c on the concave chain.
Edge e: the first (t-edge) hit by $\overrightarrow{av}$
Data structure:
(H, b_θ): the convex hull and the point
having a tangent of slope θ (figure 8)
Description:
Uses Graham scan [22] to maintain the convex hull
of endpoints of triangulation edges (t-edges)
Operations:
Add$(p, \text{f or b})$: Add points to front or back of H
ChangeSlope(θ): Change slope to θ and recalculate b_θ
Initialize
$a \leftarrow u$, $c \leftarrow next(a)$, and $c' \leftarrow next(c)$
$e \leftarrow$ the 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)
$b_\theta \leftarrow$ the last point **Add**()ed to H
repeat
ChangeSlope(slope of $\overline{ac}$)
move a along $\overline{uv}$, rotating $\overline{ac}$ around c until
case 1: if $\overline{ac}$ is part of $\overline{u'v'}$
use $\overline{ac}$ as the last link in the path
exit program
case 2: a reaches v.
use $\overline{vc}$ in the minimum-link path
exit loop
case 3: Line segment $\overline{ac}$ hits b_θ
use $\overline{ac}$ in the minimum-link path
exit loop
case 4: slope of $\overline{ac}$ points into H at b_θ
ChangeSlope(slope of $\overline{ac}$)
case 5: a hits e
if an endpoint 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
if $c \neq u'$, repeat program using $ext(\overline{ac})$ as $\overline{uv}$

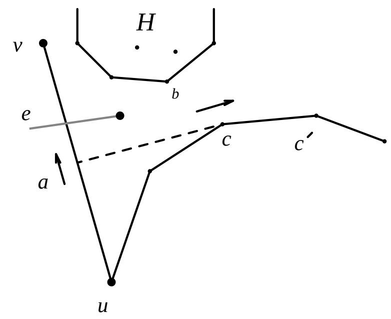


Figure 8: The hull H

Table 2: Computing the minimum-link path between inflection edges $\overline{uv}$ and $\overline{u'v'}$

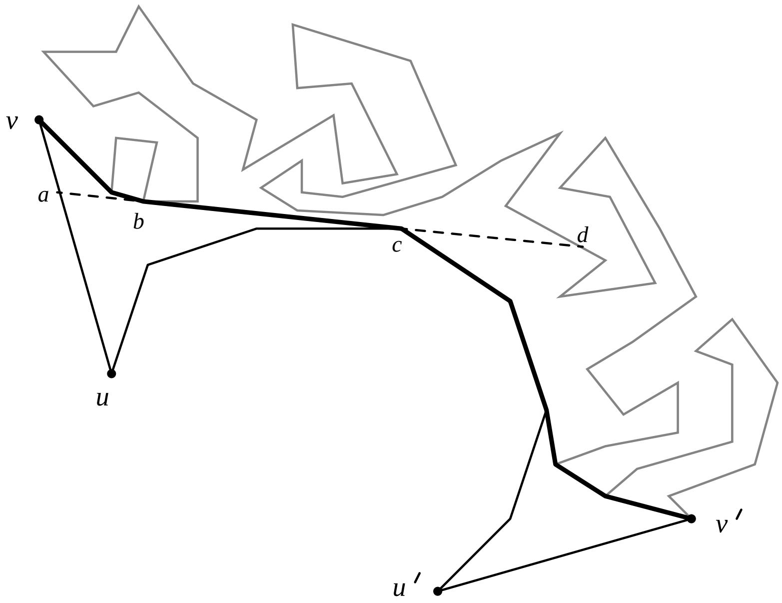


Figure 7: The path γ has inflection edge $\overline{bc}$

1. The tangent $\overrightarrow{ac}$ becomes a segment of $\overline{u'v'}$: no extra segments are needed.
2. The moving point a reaches the polygon boundary, which implies $a = v$: the segment $\overline{vc}$ is the inflection edge of γ.
3. The tangent $\overrightarrow{ac}$ 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:

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 angles, 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 9, the lemma holds. ■

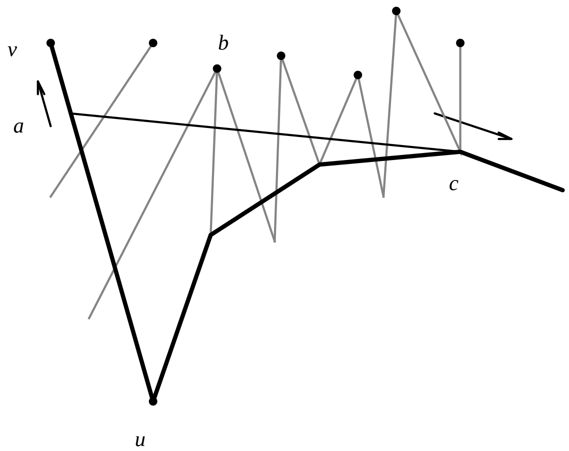


Figure 9: The sweep stops at b

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 [22] 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.

These arguments establish the correctness of the algorithm outlined in table 2. 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 9. 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 4.4.

Theorem 4.4 *A minimum-link path α', homotopic to α, can be computed in space and time proportional to $O(C_\alpha + \Delta_\alpha + \Delta_{\alpha'})$.*

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 path with k segments can intersect $\Theta(kn)$ triangulation edges, as shown in figure 10.

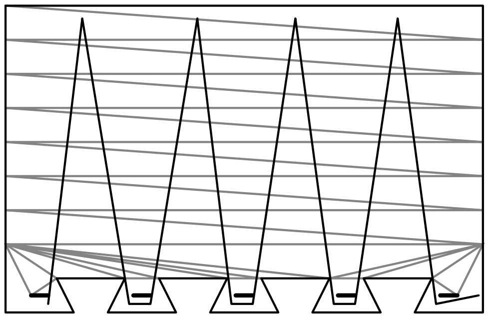


Figure 10: The k-link minimum path intersects $\Theta(kn)$ edges

5 Loops

The algorithms of the previous section can be used to find the shortest and minimum-link closed path whose starting and ending points coincide—that is, for a loop that is pinned to the starting point. For completeness, we show how to use the universal cover to help find shortest and minimum-link loops of a given homotopy class that is not pinned to pass through any given point. We compute Euclidean shortest loops in section 5.1 by simply walking around the loop at most four times; this can be applied to compute relative convex hulls [12, 51, 53] and minimum-perimeter inpolygons [11]. For minimum-link loops, section 5.2, we have nothing new to add except the obvious generalizations from nested polygons to BTMs.

5.1 Euclidean shortest loops

The funnel algorithm, outlined in section 3.1, computed a shortest path in a *sleeve* polygon—a triangulated polygon whose dual was a path. For shortest loops, we define a *band* analogously as a BTM whose dual is a single cycle. In this section, we first reduce the problem of computing the shortest loop of a given homotopy type to the problem of finding the shortest loop around a *band*.

A band is *orientable* if and only if the boundaries of its triangles can be be traversed so that each internal edge is traversed once in each direction. Orientable bands have two boundary cycles and non-orientable bands have only one. Subsections 5.1.1 and 5.1.2 deal with the orientable and non-orientable cases of the reduced problem.

Lemma 5.1 *In a BTM M with a loop α, we can compute a band whose shortest loop is the lift of the shortest loop homotopic to α in M. Computation time and space is $O(\Delta_\alpha)$.*

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, as shown in figure 11. Remove all but the last of these common triangles, and glue those together. The result is either a single triangle, in which case α is homotopic to a point, or else it is a 2-manifold M' whose dual has a single cycle—M' is a band. We must show that the band M' contains the lift of the shortest loop homotopic to α.

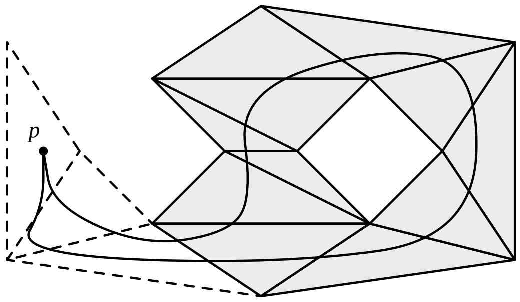


Figure 11: Constructing a band

We can begin with the band and perform the universal cover construction to obtain a 2-manifold of genus one that contains the lift of α. Suppose we remove an edge from this manifold. We either separate the manifold or, if the edge is an internal edge of the band, we reduce the genus to zero. This proves that edges internal to the band must be crossed an odd number of times and all other edges must be crossed an even number of time by any loop homotopic to α. But, just as in lemma 3.1, this implies that the shortest path crosses internal edges once and no other edges. Thus, the band contains the shortest loop homotopic to α. ∎

Before we solve the problem of computing the shortest loop around a band, we define the concepts of *turn angles* and *cut manifolds*.

The *turn angle* (figure 12) of an oriented piecewise-linear path with given starting and ending points in a BTM M is measured by following the orientation of the path and summing the angles of its turns. Each turn has an angle $-\pi < \theta < \pi$; (locally) right turns are negative and left are positive. The *turn angle of a loop* is the turn angle of the path around the loop starting and ending at the orientation of some edge—which edge is chosen does not affect the angle.

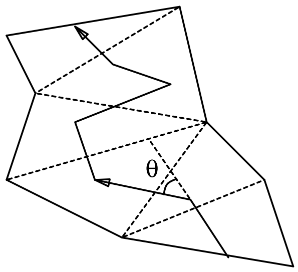


Figure 12: Turn angle

If we cut a band M along any non-boundary triangulation edge e, we obtain a simply connected manifold M_{cut} whose boundary has two copies of e. The shortest loop around M becomes a shortest path in M_{cut} between two copies of a point $p \in e$. (Czyzowicz et al. [11] show how to use shortest path maps to compute the shortest path between the two copies of e in linear time—we will use somewhat lighter artillery.) Around the boundary of M_{cut}, the copies of e have the same or opposite orientations, depending on whether the band M was orientable or non-orientable. We will handle these cases separately in the following two subsections.

5.1.1 Orientable bands

In this section, we show how to find the shortest loop around an orientable band. After defining the *inner boundary* of the band, we state a procedure using the funnel algorithm [33] to compute the shortest loop by walking around the inner boundary twice. We prove its correctness in the rest of the section.

The boundary of an orientable band M consists of two closed curves, σ_{R} to the right and σ_{L} to the left of M's cycle. According to the next lemma, the turn angle of the shortest loop in an orientable band equals the turn angle of the canonical loop or either boundary curve.

Lemma 5.2 *In an orientable BTM, two simple (i.e., non-self-intersecting) oriented loops that are homotopic have the same turn angle.*

If the turn angle of M is negative, then we say that σ_{R} is the *inner boundary*, otherwise σ_{L} is the inner boundary. In the figures, the triangles are laid out flat in the plane, which would give turn angles of $\pm 2\pi$. Manifolds that cannot be embedded in the plane give rise to other turn angles.

The following procedure computes the shortest loop:

1. Let $\overline{uv}$ be a line segment of the inner boundary.
2. Use the funnel algorithm to compute the shortest path α from u to v that winds around the band twice. (See figure 13.)
3. Let p be a vertex that appears twice on the path; the path from p to p is the shortest loop.

This algorithm is based on the fact that once we identify a point p on a shortest loop, we can compute the loop by computing the shortest path from p back around to p. Lemma 5.3 says that there is a shortest loop touching a vertex of the inner boundary.

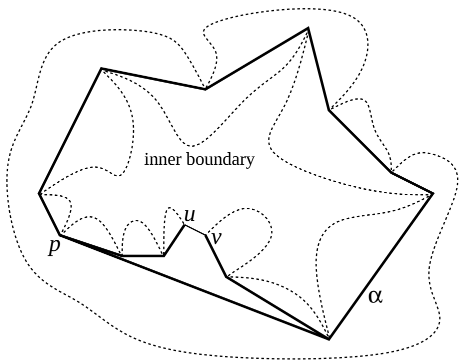


Figure 13: Around the inner boundary

Lemma 5.3 *There is a shortest loop that touches a vertex of the inner boundary.*

Proof: If the turn angle of a band M is positive, then the shortest loop must make a left turn. It can only do so by turning at a vertex of the left or inner boundary. The case of a negative turn angle is symmetric.

If the turn angle of the band M is zero then any shortest loop turns as much to the right as to the left. Thus, if it turns at all, it turns at vertices of both the inner and outer boundaries. If the shortest loop does not turn, then cut the band M along a triangulation edge e—the two copies of e are parallel and the shortest loop becomes a straight line segment ℓ between corresponding points of the copies of e. Without changing the length of the segment ℓ, one can translate ℓ to the left until it touches a vertex of the inner boundary. ∎

With this lemma, we can prove correctness.

Theorem 5.4 *Given an orientable band M composed of n triangles, the procedure above correctly computes the shortest loop around M in linear time.*

Proof: Let p be the vertex on the inner boundary of some shortest loop whose existence is proved by lemma 5.3. The shortest path λ from u starts on or inside this shortest loop and reaches p before going completely around the band. Similarly, the shortest path from v reaches p before going around the band in the other direction. Thus, p is reached twice.

The path α can thus be decomposed into three pieces: the shortest path from u to p, denoted α_u; the shortest loop around the band, denoted λ; and the shortest path from p to v, denoted α_v. The vertices of λ are obviously the vertices of the shortest loop. Together α_u and α_v compose the shortest path from u around to v—a vertex appears on this path only once. Thus, any vertex that appears twice on α is on the shortest loop and can be used in place of p. ∎

5.1.2 Non-orientable bands

One might think that computing the shortest loop in a non-orientable band would be more difficult. In this section, however, we show how to find the shortest loop that winds twice around the band by a reduction to an orientable band. We then show how to obtain the shortest loop from this curve. The result is theorem 5.5.

Theorem 5.5 *Given a non-orientable band M composed of n triangles, one can compute the shortest loop around M in linear time.*

We can conceptually take two copies of M_{cut}, reverse one left-to-right, and paste them into a single band M_{double}, as shown in figure 14. The band M_{double} is orientable and has turn angle zero: starting from triangulation edge e, you travel through one copy of M_{cut} until you encounter the reversed copy, denoted e_R. Then you travel through the reversed copy of M_{cut} until you reach e again. The turn angles in each copy of M_{cut} have opposite sign. We can use the procedure of the previous section to find the shortest loop in M_{double} that touches the left boundary—call it λ. Notice that λ is the shortest loop that winds around

M twice, so its length is at most double the length of the shortest loop in M. We shall see that the length is exactly double.

Suppose λ intersects e at a point p. Then the shortest loop touching the right boundary is the shortest path starting and ending at the corresponding point $p_R \in e_R$. In other words, the shortest loop in M_{double} touching the right boundary is λ_R—the loop λ viewed from the perspective of edge e_R. This should not be surprising as M has only one boundary.

We now consider two cases depicted in figure 14. First, if the shortest loop λ in M_{double} makes any turns, then λ makes turns on vertices of both the right and left boundaries. Since the shortest loop touching a given boundary is unique, both loops λ and λ_R are identical. Therefore, λ passes through the point $p_R \in e_R$—that is, λ winds around the shortest loop in M twice.

Second, if the shortest loop λ makes no turns, then by cutting the manifold M_{double} along e, we see that the loops touching the left and right boundaries, λ and λ_R, form two parallel lines. If the intersections with e are points p on the left and q on the right, as shown in figure 14b, then the intersections with e_R are the corresponding points q_R on the left and p_R on the right. The line λ' parallel to λ and λ_R and passing through the midpoint of the segment $\overline{pq}$ is also a shortest loop in M_{double}. Moreover, λ' also passes through the midpoint of $\overline{q_R p_R}$. But these two midpoints are just the corresponding points on two copies of e. As a result, λ' winds around the shortest loop in M twice.

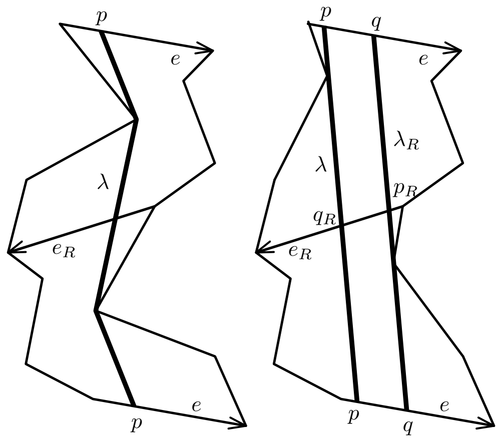


Figure 14: Cases for the shortest loop in M_{double}

5.2 Minimum-link loops

As in section 5.1, if we know a vertex or edge of a minimum-link loop, we can use the path algorithm to compute it. When a minimum-link loop is convex, however, it seems difficult to find such a vertex or edge.

Because of the algorithm of section 5.1, we can assume that our loop 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 2 to find the paths between inflection edges—a fact that has also been noted by Ghosh and Maheshwari [20]. Lemma 4.1 implies that the resulting loop is a minimum-link curve.

If α has no inflection edges, then all minimum-link loops are convex. One can use the technique of Aggarwal et al. [1] as extended by Wang [54] and Ghosh [19] to find a minimum-link loop. Briefly, one finds an initial loop finding a minimum-link path from p around to p; the resulting path has at most one segment more than the minimum. One then rotates this loop, keeping track of its points of contact with the inner and outer chains, to see if one can shorten the loop. The algorithm finds a minimum-link loop with k line segments in $O(n \log k)$ time. It would be interesting to discover a matching lower bound.

6 Paths with restricted orientations

For some applications, such as VLSI, the paths are restricted to c fixed directions; 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 6.1, we define convex polygonal distance functions appropriate to a given set of orientations. Then we show in section 6.2 that the length, under a convex distance function, of the Euclidean shortest path computed in section 3.1 equals the length of the shortest c-oriented path. Section 6.3 shows how to modify the minimum-link algorithm of section 4 to compute minimum-link c-oriented paths. Finally, section 6.4 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 [13] has independently noted this fact for rectilinear paths in simple polygons.

In each 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.

6.1 Metrics versus distance functions

When paths are restricted, the link metric remains the number of line segment of a path. We can replace the Euclidean metric, however, by a distance function that gives the length of the shortest restricted path between two points. The Manhattan or L_1 metric, 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, is an example of a distance function for rectilinear paths.

More generally, we can use Minkowski's convex distance functions [6]. 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, $\|v\|_A = \inf\{\lambda \geq 0 : v \in \lambda A\}$. The distance from point r to s is $\|s-r\|_A$. The distance function need not induce a metric because it need not be symmetric: $\|v\|_A$ may not equal $\|-v\|_A$. It does, however, satisfy the triangle inequality [6]: 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 $\|\cdot\|_A$. Choosing A to be the unit circle gives the Euclidean metric; choosing A to be the diamond defined by the four unit vectors in the axial directions gives the L_1 metric. For a c-oriented path, which is a path restricted to follow the orientations of c unit-length *basis vectors* $u_1, u_2, \ldots, u_c$, we choose A to be the convex hull of $\{\vec{0}, u_1, \ldots, u_c\}$. We assume that the u_i appear on A in the order listed.

As an aside, if the origin $\vec{0}$ is on the boundary of A then vectors that are not contained in the angle formed by the boundary of A at $\vec{0}$ have infinite length. They cannot be reached by a c-oriented path because they cannot be expressed as a positive linear combination of the basis vectors.

A path that follows c chosen orientations has the same length under the Euclidean metric and under the associated convex distance function. More importantly, a vector v measured under a convex distance function has the length of the shortest c-oriented path from the origin to v—we show this in the next lemma.

Lemma 6.1 *Let A be the convex hull of $\{u_1, u_2, \ldots, u_c\}$, a circularly-ordered set of basis vectors and let v be a vector in the wedge defined by adjacent basis vectores u_i and u_{i+1}. Vector $v = au_i + bu_{i+1}$ iff $\|v\|_A = a+b$.*

Proof: This is true for the unit vectors of the distance function $\|v\|_A = 1$, which are on segments, $\alpha u_i + (1-\alpha)u_{i+1}$ for $0 \leq \alpha \leq 1$, that join adjacent basis vectors on the boundary of the convex hull. Since length scales with the vector, it remains true for arbitrary vectors v. ∎

6.2 Shortest paths under a convex distance function

We use lemma 6.1 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 3.2 use only the triangle inequality to show that the path computed in section 3.1 is minimum under the Euclidean metric. But this implies that

Theorem 6.2 *The Euclidean shortest path computed in section 3.1 is a shortest path under any convex distance function.*

Lemma 6.1 implies that if we replace each segment of the Euclidean shortest path by a "zig-zag" or "staircase" made from the two adjacent basis vectors, then we will have a c-oriented path of the same (minimum) length. By cutting the Euclidean shortest path at all points with tangent vectors that are among the c basis vectors and computing c-oriented "staircases" for the resulting pieces we will find a shortest c-oriented path that has the fewest possible links.

Lemma 6.3 *Let t be a point of the Euclidean shortest path α having a basis vector u 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 segment through t and parallel to u. Any path from p to q first crosses this segment at or before t and last crosses it at or after t. By applying lemma 6.1 to the wedge containing u alone, we see that the shortest path under the convex distance function is inside this segment from the first to last crossing and therefore passes through t. ■

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 [24]—and cutting it whenever the direction of travel is one of the c basis vectors. (If c is not considered a constant, then we can use binary search to find the wedges that contain the slopes of edges. The time complexity becomes $O(C_\alpha \log c)$.)

The slopes on each resulting path lie in a wedge defined by two adjacent basis vectors. Thus, by lemma 6.1, each path should be replaced by a path using only those two orientations. If we are unconcerned about the number of links then, using the two orientations, we can remain within an arbitrarily small neighborhood of the Euclidean shortest path. More likely, however, one would want to construct a shortest c-oriented path using the smallest number of links. The next section develops an algorithm for the more general problem of computing minimum-link c-oriented paths. Section 6.4 mentions how this algorithm can be simplified when there are only two directions of interest and also discusses when a minimum link path can also be a shortest path under the convex distance function.

6.3 Minimum-link c-oriented paths

This section develops a greedy algorithm to compute a minimum-link c-oriented path homotopic to a path α from p to q: Each link (line segment) reaches as far as possible towards q, guided by the Euclidean shortest path. Define the i-link region of p, denoted $R_i(p)$, to be the set of all points of the universal cover that can be reached by a c-oriented path of i links from p. If q is not in $R_i(p)$ then we shall find two c-oriented segments on the boundary $\partial(R_i(p))$ with adjacent orientations that separate p from q in the universal cover. We compute the two segments of $\partial(R_{i+1}(p))$ that separate p from q from the two segments of $\partial(R_i(p))$.

Section 4 used essentially this greedy approach to compute an unrestricted minimum-link path. In that case, however, the boundary of the i-link region is a single line segment and one can compute the links of the path in time proportional to the number of triangles that the path crosses. In this section we will have to explore two possible ways to reach the goal q. This difficulty arises already in rectilinear paths, where one must decide whether to begin with a horizontal or a vertical step. (It is interesting that, even with more allowed orientations, no more than two paths need be considered at any time.) The time required by our algorithm will therefore be proportional to the number of triangles explored, which may be greater than the number of triangles intersected by the final path. The worst-case bounds are similar to those of section 4: 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 basis vector directions are initially sorted, then the algorithm can be implemented to run in $O(nk \log c)$ and $O(n + k \log c)$ time, respectively.

Lemma 6.4 *Let α be a Euclidean shortest path from p to q in a BTM M, and let U be the universal covering space of M. Suppose Q is the connected component of $U - R_i(p)$ that contains q. Then $\partial(Q) - \partial(U)$ consists of at most two segments from a point r that have adjacent orientations u and v.*

Proof: We prove this by induction. The 1-link region $R_1(p)$ consists of maximal length segments radiating out from p in the permitted orientations. If we cut the universal cover U along these segments and look at the component Q that contains q, we find that $\partial(Q)$ contains a portion of one of these segments in degenerate cases or two adjacent segments meeting at p in non-degenerate cases.

Now, assume that the i-link region has boundary segments r_u and r_v that come from the point r in directions u and v, as depicted in figure 15. Without loss of generality, we can assume that u is clockwise of v and that the Euclidean path α turns to the right. Any minimum-link c-oriented path ρ that is homotopic to α must cross one of the segments r_u or r_v; by cutting the path ρ at this crossing point and replacing the initial portion with an i link path, we can assume that ρ uses part of r_u or r_v as its ith link.

There are two cases to consider when extending the path ρ by one link, depending on whether there is some c-oriented segment from r_u or r_v that is tangent to the Euclidean shortest path α or not.

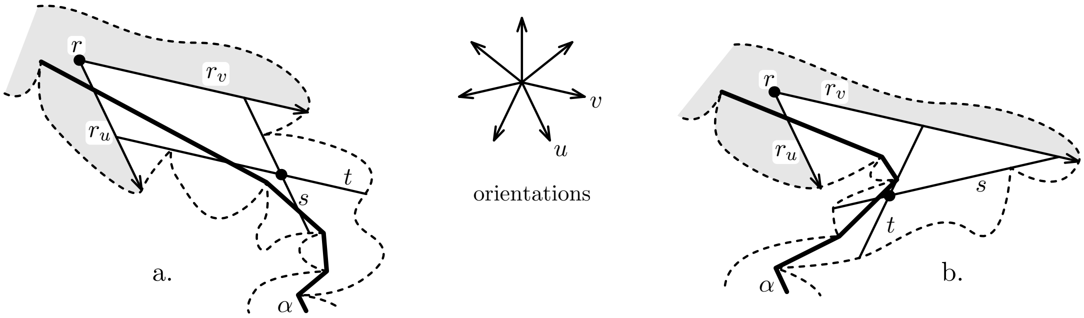


Figure 15: Computing part of $\partial(R_{i+1}(p))$ from $\partial(R_i(p))$, shaded

Suppose, first, that there is no such tangent, as in figure 15a. Imagine drawing maximal length segments in Q from r_v in direction u; there will be some segment s that that is the last one crossed by α en route to q. Similarly, draw the segment t from r_u in direction v that is last crossed by α.

We notice that segments s and t intersect: If α hits s after t, as in figure 15a, then r_u, r_v, s and α bound a simply connected region in U that t enters by crossing α. Since t cannot cross r_u or r_v or recross α, t must cross s. Let $r' = s \cap t$. The $(i+1)$-link region is bounded by the portions of s and t in the directions of u and v from r' because they are the extremal segments in the directions of u and v and segments in other orientations cannot reach from r_u or r_v to the segments from r'. This establishes the lemma for the first case.

Second, suppose that there is a c-oriented tangent to α from r_v. Let t be the c-oriented tangent furthest clockwise from v, as shown in figure 15b. Tangent $t \subset U$ has an end on r_v, is tangent to α at b, and has maximal length. Now, draw segments from r_v in the next orientation clockwise from the orientation of t and let s be the last segment crossed by α. Let $r' = s \cap t$. If r' lies between r_v and b on t, then the $(i+1)$-link region is bounded by the segment of t following b. Otherwise, r' follows b on t and the $(i+1)$-link region is bounded by the two segments from r' in the directions of s and t. ∎

This lemma and its proof indicate a way to start from p and obtain a sequence of points, each of which a minimum-link c-oriented path can pass through in one of two adjacent directions. If we can compute these points and directions, then we can construct the minimum-link path as follows: Begin at $r = p$ with the at most two candidate paths whose initial orientations delimit the smallest wedge containing the orientation of the first segment of the Euclidean shortest path α. Obtain the next point r' and the pair of directions, which come from non-tangents in the first case of lemma 6.4 and from a tangent and a non-tangent in the second case. In the first case, both candidate paths are extended by one link. In the second, the candidate path that cannot be extended by a tangent is discarded and the path up through r is fixed. Then new candidate paths are begun that pass through r' in two directions and the process continues. The algorithm stops with a minimum-link c-oriented path when one of the candidate paths reaches q—a fact that can be detected by the tangent-finding algorithm.

To compute the sequence of points and directions we need to find extreme c-oriented tangents to α, if they exist, and find extreme segments of fixed orientations that connect the old link region boundary to α. Both of these tasks can be performed by a modification of the minimum-link path algorithm presented in table 2. Since this algorithm finds the extreme tangent by sweeping a tangent segment and recording in a convex hull the endpoints of triangulation edges that cross the sweep, there is little modification required to find an extreme c-oriented tangent. In figure 15a, the sweep would begin at the intersection of α and r_u and maintain a tangent segment to α as the other endpoint moved up to r and then along r_v. When the extreme tangent is found, the extreme c-oriented tangent, if any, can be reported by searching the list of orientations.

The same idea—sweeping a segment and maintaining the endpoints of triangulation edges that cross the sweep—applies to find the extreme segment with a given orientation. Since the orientation is fixed, one does not need to record the convex hull of the endpoints ahead of the sweep. Storing the first endpoint that will be encountered is sufficient.

The extra exploring means that we do not have output-sensitive bounds for c-oriented paths.

Theorem 6.5 *One can construct a minimum-link c-oriented path homotopic to α in time and space proportional to the number of triangles that contain candidate segments for the minimum-link path.*

If the c allowed orientations are initially sorted, then the worst-case time bound to compute a k-link path in a BTM with n triangles is $O(nk \log c)$. This analysis can be sharpened for a simply connected BTM if there are two opposite basis vectors. If we use the algorithm of Fournier and Montuno [17] 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. Any trapezoid edge that intersected more than three edges of a path could be used to shorten the path. Thus, each trapezoid is explored a constant number of times. The running time of the algorithm in this case is $O(n + k \log c)$.

6.4 Simultaneous minimization of length and links

In the c-oriented case, as in the unrestricted case, a minimum-link path is usually not the shortest path and vice versa. A "long" straight detour can generally save several turns. 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 6.2 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. But this breaking implies that only the first case for extending a path can arise in lemma 6.4. Therefore, we can compute such paths entirely by sweeping segments with fixed orientations—we need not maintain convex hulls to determine where the sweep ends. We do need to try both candidate paths, however, and merge collinear segments from separate pieces to 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 B satisfies the *halfplane condition* if there is a halfplane that contains all of B 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 [24], and noting in which basis vector orientations you face during a turn about a vertex. If the basis vector tangent at a vertex has the halfplane condition, then shortest and minimum-link paths can both pass through that vertex in the direction of the basis vector.

Lemma 6.6 *If a tangent basis vector, u, 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 u.*

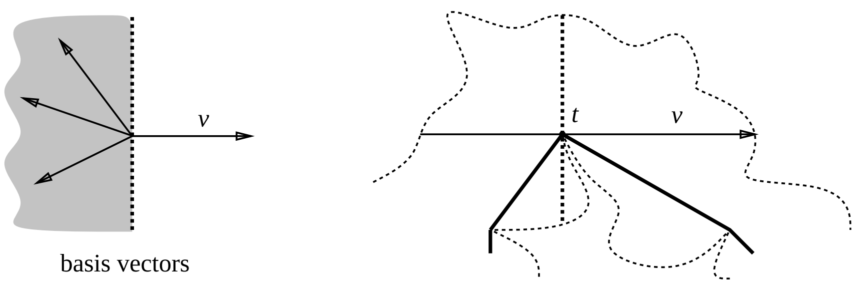


Figure 16: The halfplane condition

Proof: Cut the universal cover through t along a line that defines a halfplane containing all the basis vectors except u, as illustrated in figure 16. If any c-oriented path crosses this cut at t, then it must do so in direction u by the halfplane condition and the fact that u is a tangent 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 u. Separate the universal cover into three pieces by a line segment through t and parallel to u. 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 6.6 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.

7 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 new algorithms for computing minimum length and minimum link c-oriented paths.

Acknowledgements

We would like to thank Joseph Mitchell and Leonidas Guibas for discussions on these and related problems, and the referees for their comments. This research has been supported by Digital Equipment Corporation, ESPRIT Basic Research Action No. 3075 (project ALCOM) and an NSERC Research Grant (Canada).

References

[1] A. Aggarwal, H. Booth, J. O'Rourke, S. Suri, and C. K. Yap. Finding minimal convex nested polygons. *Information and Computation*, 83(1):98–110, Oct. 1989.

[2] E. M. Arkin, J. S. B. Mitchell, and C. D. Piatko. Bicriteria shortest path problems in the plane. In *Proceedings of the Third Canadian Conference on Computational Geometry*, pages 153–156, Simon Fraser University, Vancouver, 1991.

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

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

[5] B. Bhattacharya and G. T. Toussaint. A linear algorithm for determining translation separability of two simple polygons. Technical Report SOCS–86.1, School of Computer Science, McGill University, Montreal, 1986.

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

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

[8] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6:485–524, 1991.

[9] 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.

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

[11] J. Czyzowicz, P. Egyed, H. Everett, D. Rappaport, T. Shermer, D. Souvaine, G. Toussaint, and J. Urrutia. The aquarium keeper's problem. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 459–464, Jan. 1991.

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

[13] M. de Berg. On rectilinear link distance. *Computational Geometry: Theory and Applications*, 1:13–34, 1991.

[14] M. de Berg, M. van Kreveld, M. Overmars, and B. Nilsson. Finding shortest paths in the presence of orthogonal obstacles using a combined L_1 and link metric. In *SWAT 90: Second Scandinavian Workshop on Algorithm Theory*, number 447 in Lecture Notes in Computer Science, pages 211–224. Springer-Verlag, 1990.

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

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

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

[18] S. Gao, M. Jerrum, M. Kaufmann, K. Mehlhorn, W. Rülling, and C. Storb. On continuous homotopic one layer routing. In *Proceedings of the Third Annual ACM Symposium on Computational Geometry*, pages 392–402, 1987.

[19] S. K. Ghosh. Computing the visibility polygon from a convex set and related problems. *Journal of Algorithms*, 12:75–95, 1991.

[20] S. K. Ghosh and A. Maheshwari. An optimal algorithm for computing a minimum nested nonconvex polygon. *Information Processing Letters*, 36:277–280, 1990.

[21] S. K. Ghosh and D. M. Mount. An output sensitive algorithm for computing visibility graphs. *SIAM Journal on Computing*, 20(5):888–910, 1991.

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

[23] 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.

[24] 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.

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

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

[27] L. J. Guibas, J. E. Hershberger, J. S. B. Mitchell, and J. S. Snoeyink. Minimum-link approximation of polygons and subdivisions. In W. L. Hsu and R. C. T. Lee, editors, *ISA '91 Algorithms*, number 557 in Lecture Notes in Computer Science, pages 151–162. Springer-Verlag, 1991.

[28] V. W. Guillemin and A. Pollack. *Differential Topology.* Prentice Hall, Englewood Cliffs, NJ, 1974.

[29] R. H. Güting. *Conquering Contours: Efficient Algorithms for Computational Geometry.* PhD thesis, Dortmund, 1983.

[30] R. H. Gütting and T. Ottmann. New algorithms for special cases of hidden line elimination problem. *Computer Vision, Graphics, and Image Processing*, 40:188–204, 1987.

[31] 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.

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

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

[34] C. E. Leiserson and F. M. 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.

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

[36] J. S. B. Mitchell, C. Piatko, and E. M. Arkin. Computing a shortest k-link path in a polygon. In *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science*, pages 573–582, 1992.

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

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

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

[40] B. Nilsson, T. Ottmann, S. Schuierer, and C. Icking. Restricted orientation computational geometry. In *Data structures and efficient algorithms.*, number 594 in Lecture Notes in Computer Science, pages 148–185. Springer-Verlag, 1992.

[41] B. O'Neill. *Elementary Differential Geometry.* Academic Press, New York, 1966.

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

[43] F. P. Preparata and M. I. Shamos. *Computational Geometry—An Introduction.* Springer-Verlag, New York, 1985.

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

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

[46] J.-R. Sack. *Rectilinear Computational Geometry.* PhD thesis, McGill University, 1984.

[47] M. Sharir and A. Schorr. On shortest paths in polyhedral spaces. *SIAM Journal on Computing*, 15:193–215, 1986.

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

[49] S. Suri and J. O'Rourke. Finding minimal nested polygons. In *Proceedings of the 23rd Annual Allerton Conference on Communication, Control and Computing*, pages 470–479, 1985.

[50] X.-H. Tan, T. Hirata, and Y. Inagaki. The intersection searching problem for c-oriented polygons. *Information Processing Letters*, 37:201–204, 1991.

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

[52] G. T. Toussaint. Movable separability of sets. In G. T. Toussaint, editor, *Computational Geometry*, pages 335–376. North Holland, Amsterdam, 1985.

[53] G. 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.

[54] C. A. Wang. Finding minimal nested polygons. *BIT*, 31:230–236, 1991.

[55] C. A. Wang and E. 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.