

Testing Homotopy for Paths in the Plane*

Sergio Cabello
Inst Info & Comp Sci
Utrecht University
the Netherlands
sergio@cs.uu.nl

Yuanxin Liu
Dept Comp Sci
UNC Chapel Hill
NC, USA
liuy@cs.unc.edu

Andrea Mantler
Dept Comp Sci
UNC Chapel Hill
NC, USA
mantler@cs.unc.edu

Jack Snoeyink
Dept Comp Sci
UNC Chapel Hill
NC, USA
snoeyink@cs.unc.edu

ABSTRACT

In this paper we present an efficient algorithm to test if two given paths are homotopic; that is, whether they wind around obstacles in the plane in the same way. For simple paths specified by n line segments with obstacles described by n points, our algorithm runs in $O(n \log n)$ time, which we show is tight. For self-intersecting paths the problem is related to Hopcroft's problem.

Categories and Subject Descriptors

F.2.2 [Analysis of Algorithms]: Nonnumerical Algorithms—
Geometrical problems and computations

General Terms

Algorithms

Keywords

Homotopy, Computational topology, Simple paths

1. INTRODUCTION

A basic topological question is determining if two paths are homotopic, so that one can be deformed into another without leaving the containing space. Specifically, suppose that the input consists of a set P of up to n points in the plane, and two paths, α and β , that start and end at the same points and are represented as polygonal lines of at most n segments each. The goal is to determine whether α is deformable to β without passing over any points of P . Equivalently, we determine whether the closed loop $\alpha\beta^R$, which concatenates α with the reverse of β , is contractible in the plane minus P . We assume (or simulate) general position, so that no three points are colinear and no two points

*The first author is partially supported by the Cornelis Lely Stichting, and second and third by NSF grants 9988742 and 0076984.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SoCG'02, June 5–7, 2002, Barcelona, Spain.

Copyright 2002 ACM 1-58113-504-1/02/0006 ...\$5.00.

are on the same vertical line. We are primarily concerned with paths that have no self-intersections.

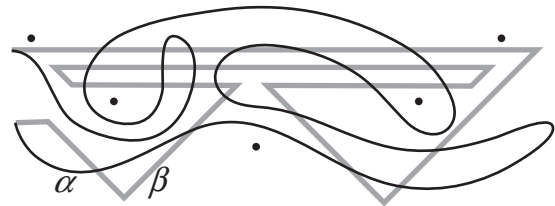


Figure 1: Are α and β homotopic?

The path homotopy question arises in several application areas: In circuit board design, *river routing*, where the homotopy class of each wire is specified, is one of the few polynomial-time solvable variations of the wire routing problem [18]. In motion path planning, one might check to see that two ways of getting from point A to point B are equivalent [16]. In geographic information systems (GIS), one may wish to simplify a linear feature (road or river) while respecting the way in which the feature winds around points [4, 7]. Michael Goodchild, in an invited lecture at the 11th ACM Symposium on Computational Geometry, pointed out that even on a road map that has features with 60m accuracy, you will still find all the houses on the proper side of the road. In such a case, the operators entering data have used topological constraints to make sure that the road winds properly when creating the road layer or building layer.

Several approaches to test path homotopy among points in the plane give algorithms with quadratic worst-case behavior. One approach is to find the Euclidean shortest representatives of the homotopy classes for α and β using known algorithms, then to check that they are identical. These algorithms [16] trace each path through a tri-

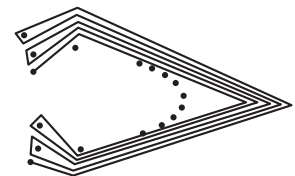


Figure 2: $\Theta(n^2)$ segments in shortest homotopic path

angulation, taking time proportional to the number triangles that intersect the paths, which may be $\Theta(n^2)$. In fact, explicit representation of the shortest path may take $\Theta(n^2)$ segments for a path like that of Figure 2. The more general problem of testing if two paths are homotopic in compact surfaces, with or without boundary, was treated in [8, 9]. Here the specification of the path essentially must be as a

sequence of the edges that are crossed in the universal cover of the surface, which may again be quadratic.

Our approach is to convert α and β separately to near-canonical representations of their homotopy classes, then compare representations to determine if the paths are homotopic. After laying the topological groundwork in the next section, we show how to test homotopy for simple paths in Section 3. We comment on extending our algorithm for non-simple paths in Section 4, establish lower bound on both problems in Section 5. We conclude in Section 6.

2. TOPOLOGICAL PRELIMINARIES

We actually solve three natural variations of the path homotopy question, so clear definitions are important.

2.1 Three variations on path homotopy

The topological concept of *homotopy* formally captures the notion of deforming paths [1, 21]. Let $I = [0, 1]$ denote the unit interval and M denote a topological space, which for us will be the complement of some point or polygonal obstacles in the plane. A *path* is a continuous function $\alpha: I \rightarrow M$; that is, a function for which the preimage $\alpha^{-1}(A)$ of an open set $A \subseteq M$ is open in I . Paths α and β that share starting and ending points, $\alpha(0) = \beta(0)$ and $\alpha(1) = \beta(1)$, are said to be *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 $\Gamma: [0, 1] \times [0, 1] \rightarrow M$ such that $\Gamma(x, 0) = \alpha(x)$ and $\Gamma(x, 1) = \beta(x)$, and $\Gamma(0, y) = \alpha(0) = \beta(0)$ and $\Gamma(1, y) = \alpha(1) = \beta(1)$.

By the standard topological definition, path end points must lie in the space M , and the path can pass over them by continuous deformation, creating self-intersections. This may be undesirable in some applications, so we also consider two alternative definitions that allow a path to begin or end at obstacles in the plane. Informally, we can think of the path as a thread winding above a plane that is punctured with long needles that serve as obstacles. The three variations are whether a path end point is fixed by a tack, pin, or pushpin.

tack A thumbtack pushed flat into the plane presents no obstacle to the thread, so this corresponds to the standard definition given above.

pin A straight pin serves as a point obstacle, so that the end points p_0 and p_1 are not included in the topological space M . A *path* is a continuous function from an open interval $\alpha: (0, 1) \rightarrow M$ such that the one-sided limits $\lim_{x \rightarrow 0^+} \alpha(x) = p_0$ and $\lim_{x \rightarrow 1^-} \alpha(x) = p_1$.

pushpin A pushpin serves as a larger obstacle, so that closed ϵ neighborhoods around the the end points are not included in the topological space M . A *path* is defined as in the pin case, but now the limit points p_0 and p_1 are chosen on the boundaries of the ϵ neighborhoods. This is equivalent to fixing the path direction at the endpoint, or to adding point obstacles infinitesimally to the left and to the right of the path endpoint.

The two examples of Figure 3 show that these definitions lead to different notions of path homotopy. Paths α and β are homotopic under the tack definition, but not under either the pin or pushpin definitions. In fact, any homotopy will have to pass over an endpoint and create a path with

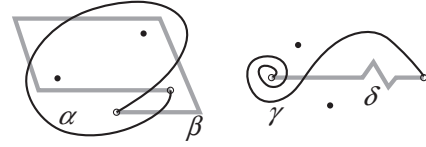


Figure 3: Distinguishing between path definitions

self-intersections. Paths γ and δ are homotopic under the tack and pin definitions, but are not homotopic under the pushpin definition, which will preserve how γ winds around the starting point.

Because we can add obstacles to the plane to handle the pin and pushpin definitions, we consider the tack (standard) definition in greatest detail. This does cause extra complications in handling self-intersections, especially in Section 3.5; we will comment in Section 3.6 on the modifications to handle the other definitions.

2.2 Canonical sequences

Whatever definition of path we choose, path homotopy is an equivalence relation [1, 21], which implies we can identify a homotopy class by giving a representative path.

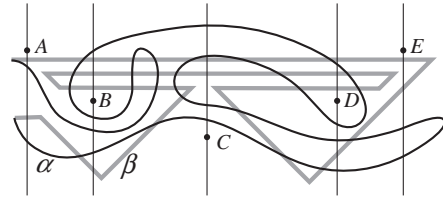


Figure 4: Paths $\alpha = \underline{A}\underline{B}\underline{B}\underline{B}\underline{C}\underline{D}\underline{D}\underline{C}\underline{C}\underline{D}\underline{E}\underline{E}\underline{D}\underline{C}\underline{B}\underline{A}$ and $\beta = \underline{A}\underline{B}\underline{C}\underline{D}\underline{E}\underline{E}\underline{D}\underline{D}\underline{D}\underline{C}\underline{B}\underline{B}\underline{B}\underline{A}$

We can write a simple representation of a path as the sequence of points that it passes above (overbar) and below (underbar). We can depict this by drawing vertical rays upward and downward from each point of P to form vertical *slabs*. The sequence just records intersections with these rays as we traverse a path through the slabs. Figure 4 illustrates the paths $\alpha = \underline{A}\underline{B}\underline{B}\underline{B}\underline{C}\underline{D}\underline{D}\underline{C}\underline{C}\underline{D}\underline{E}\underline{E}\underline{D}\underline{C}\underline{B}\underline{A}$ and $\beta = \underline{A}\underline{B}\underline{C}\underline{D}\underline{E}\underline{E}\underline{D}\underline{D}\underline{D}\underline{C}\underline{B}\underline{B}\underline{B}\underline{A}$. This representation can be constructed for paths with or without self-intersections.

A repeated point with opposite bars is a *turn point*. Our α sequence has two turn points, $\underline{B}\overline{B}$ and $\underline{D}\overline{D}$. Polygonal path α can have at most $n - 1$ turn points, since there must be at least one vertex of α between intersections with two rays from the turn point, and each vertex is claimed by at most one turn point.

An adjacent pair of repeated symbols can be deleted by deforming the curve out of a slab without changing the homotopy class. This deletion can be repeated until we obtain a *canonical sequence*. (The canonical sequence may even be empty.) For example, from α we delete $\underline{B}\overline{B}$ and $\underline{D}\overline{D}$, and from β we delete $\underline{E}\overline{E}$, $\underline{D}\overline{D}$, and $\underline{B}\overline{B}$, to find that both have the same canonical sequence $\underline{A}\underline{B}\underline{C}\underline{D}\underline{D}\underline{C}\underline{B}\underline{A}$. In fact, two paths have the same canonical sequence if and only if they are homotopic. This is most easily seen using universal covers, which we sketch in the next subsection. (Unfortunately, canonical sequences can have quadratic length, so we cannot store and manipulate them explicitly.)

2.3 Covering space

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 .

A space is always a covering space of itself under the identity map [1, 21]. A more interesting covering space is the universal covering space, which is simply connected—every loop in this space can be contracted to a point. In our setting (Figure 4), this space is most easily described by an procedure that grows a region by gluing together copies of vertical slabs at their boundary rays. Start with a region that consists of any single vertical slab, and therefore has four boundary rays (or two if we started with the leftmost or rightmost slab.) Then loop forever, selecting a boundary ray and gluing on a copy of the missing slab along that ray. Never form a cycle or enclose a ray’s endpoint.

When the set of obstacle points, P , is non-empty, the universal covering space is infinite, which is why our procedure does not terminate. It is relatively easy to construct only the portions of the universal cover that intersect the given paths α and β because any path can be *lifted* to the universal cover by choosing the sequence of rays and slabs in the order that a path intersects them. In fact, every path in the plane minus P has a unique lift into the universal cover once the starting point is specified [1, 21]. To test path homotopy, one can simply lift both α and β to the universal cover starting from the same point and ask if they end at the same point in the universal cover.

Because the universal cover is simply connected, the dual graph, with a vertex for each slab, is an infinite tree. The dual of the portion visited by a path is a finite tree, and the operation of constructing a canonical sequence prunes leaves from this tree so that what remains is the unique shortest path from the slab of the start point to the slab of the end point. This establishes the difficult-to-prove direction of the following lemma.

LEMMA 1. *Two paths have the same canonical sequence if and only if they are homotopic.*

PROOF. It is clear that two paths with the same canonical sequence are homotopic, since the construction of a canonical sequence from a path is a concatenation of homotopies on slabs. For the reverse, suppose that paths α and β are homotopic. When we lift both to the universal cover, starting at the same point, they end at the same point. Their canonical sequences, therefore, must go from the same starting slab to the same ending slab. There is only one shortest path that does so in the tree, so the canonical sequences are the same. \square

The following Corollary will be useful in proving that paths have the canonical sequence.

COROLLARY 2. *Any path that can be lifted to the universal cover to start and end on the same vertical segment has an empty canonical sequence.*

PROOF. The path is homotopic to the vertical segment, which has an empty canonical sequence. \square

3. HOMOTOPY TEST FOR SIMPLE PATHS

In this section we focus on homotopy testing for paths α and β that are *simple*—they have no self-intersections. An aboveness ordering allows us to find a rectified representation of a path. Using orthogonal range queries, we can adjust the path to have the canonical sequence, then test homotopy for two adjusted paths in $O(n \log n)$ time.

3.1 Aboveness ordering

For sets in the plane one can define an *aboveness relation* that is useful in many algorithms in computational geometry [23]: $A \succ B$ if there are points $(x, y_A) \in A$ and $(x, y_B) \in B$ with $y_A > y_B$.

We say that a set is *vertically convex* if it is path connected and the intersection with any vertical line is an (open or closed) interval. Break a path α into monotone fragments and conceptually separate them at their common endpoints to give a collection of disjoint vertically convex sets. When applied to disjoint, vertically convex sets, the aboveness relation is a partial order.

LEMMA 3. *For pairwise disjoint, vertically convex sets A_1, A_2, \dots, A_k in the plane, the aboveness relation is acyclic.*

It is easy to develop a plane sweep algorithm that can compute in $O(n \log n)$ time, a total order for monotone fragments of a path and points that is consistent with the aboveness relation. We sketch an algorithm adapted from Palazzi and Snoeyink [22].

Define an *aboveness tree* on disjoint points and monotone path fragments, in which each point and path is a child of the path directly above its rightmost endpoint. This is a k -ary tree, in which children can be ordered left to right by rightmost endpoints. We add a horizontal segment at $y = \infty$ to serve as the root of this tree. For disjoint paths specified by n total segments, this tree can be constructed in $O(n \log n)$ time by a plane sweep algorithm. We represent

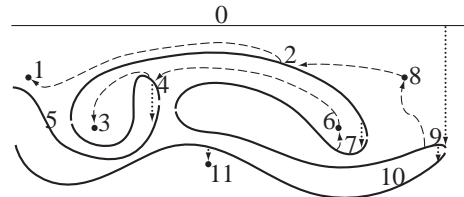


Figure 5: Inorder numbering of the child/sibling representation of the aboveness tree for path α .

this k -ary tree as a binary tree by giving each path a pointer to its left sibling and rightmost child, as in Figure 5. We then number the points and paths according to an inorder traversal that recursively visits the left sibling, the node, and then the rightmost child.

We can prove that this numbering is consistent with the aboveness relation. Define a *rightward trace* from any point or monotone path by tracing paths to their rightmost endpoints, and to parent pointers, up to the root. Note that traces do not cross. If $s \succ t$ then a trace from s must come from the left to meet a trace from t , and the inorder traversal numbers the subtree containing s before the subtree containing t . See [22] if more detail is desired.

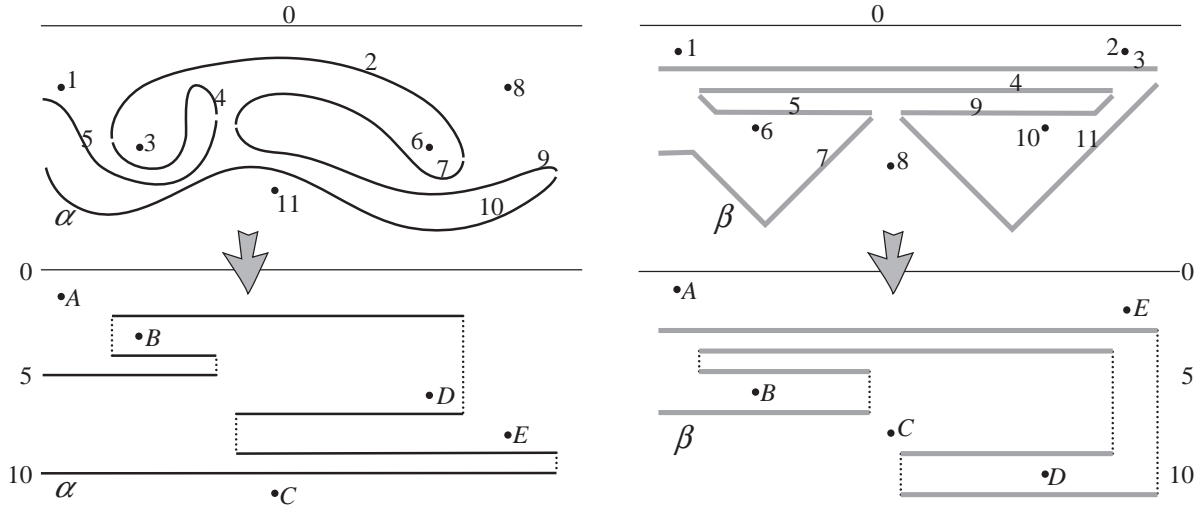


Figure 6: Numberings for α and β are used to form rectified pairs (α, P) and (β, P) . Rectifying preserves path simplicity and sequences.

3.2 Rectified pairs

We can use a total ordering that respects the aboveness to rectify any simple path α around points P . Simply rank each point of P and monotone fragment of the path and replace all y coordinates by their ranks. We call the result a *rectified pair* (α, P) .

Figure 6 shows the rectified pairs (α, P) and (β, P) . Each path is considered to consist of horizontal segments that come from monotone path fragments and vertical segments that connect these fragments in order.

Notice that the points receive different ranks for the numberings of the two paths, and therefore different y coordinates in the two rectified pairs. The points can still be distinguished by their x coordinates, which were assumed to be distinct and do not change. All aboveness relationships are preserved, so rectifying a pair does not change a path sequence or cause self-intersection.

3.3 Orthogonal range queries

Rectifying a pair makes it easier to compute a canonical sequence, because we can search for turn points using orthogonal range queries. Our problem is to preprocess a set of points P in the plane to answer queries of the form, “Report the rightmost point in an axis-aligned query rectangle Q , or ‘none’ if the rectangle Q is empty.” (We use a symmetric structure to query for leftmost points.) These are sometimes known as three-sided range queries, because it is sufficient to supply the top, bottom, and right side of the query rectangle. This problem can be solved in $O(n \log n)$ time and linear space using Chazelle’s data structure for segment dragging [5]. The RT of Edelsbrunner [11] or the range priority tree of Samet [24] achieve the same time, with an extra logarithmic factor in space but a savings in programmer complexity.

LEMMA 4. *Three-sided range queries can be answered in $O(\log n)$ time, with $O(n \log n)$ preprocessing and linear space.*

PROOF. Chazelle’s segment dragging structure achieves this bound with linear space [5]. We describe a simple structure with $O(n \log n)$ space based on persistence [10] in the

hope that it can be improved: Sort the points by increasing x coordinate, so that p_1, p_2, \dots, p_n are ordered from left to right. Denote the prefixes by $P_i = \{p_1, p_2, \dots, p_i\}$, for $1 \leq i \leq n$. Let $P_0 = \emptyset$.

Build a balanced priority search tree T_i for $i = 0$ to n . The leaves of T_i contain the points of P_i ordered by y coordinate, and every internal node indicates the point in its subtree with greatest x coordinate. We can build these incrementally using node-copying persistence. Form T_i by adding p_i to T_{i-1} : copy the nodes on a root-to-leaf path, and modify the copies to preserve balance and update priority. This takes a total of $O(n \log n)$ time and space.

To answer a query for the rightmost point in Q , simply search on x to find the tree T_i that is right of the line through the right side of Q , search in T_i for the subtrees between the top and bottom sides of Q , then find the maximum x in all these subtrees. This takes $O(\log n)$ time. \square

3.4 A rectified canonical path

Given a rectified pair (α, P) , we can compute a new path whose sequence is the canonical sequence for α . For ease of expression, we call this new path a *canonical path*. It must be understood that this is defined with respect to a rectified pair, and not the original points. We show how to compare canonical paths for (α, P) and (β, P) in the next subsection.

We can use an incremental algorithm to compute a canonical path from the rectified pair (α, P) . We assume that α is represented by the list $x_0, y_0, x_1, y_1, \dots, y_{n-1}, x_n$, which indicates that the path visits $(x_0, y_0), (x_1, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1}), (x_n, y_{n-1})$, alternating between horizontal and vertical segments.

We consider the horizontal segments of α in order. We maintain the invariant that after i segments, for $0 \leq i \leq n$, stack S contains a canonical path starting from x_0 and ending with x_i . That is, the canonical sequence for the path in S has no adjacent repeated symbols.

Begin by setting $i = 0$ and pushing x_0 onto S . This establishes the initial stack invariant.

While $i < n$, let X denote the top element on the stack, and if $X \neq x_0$, let Y_s and X_s be the pair just below X .

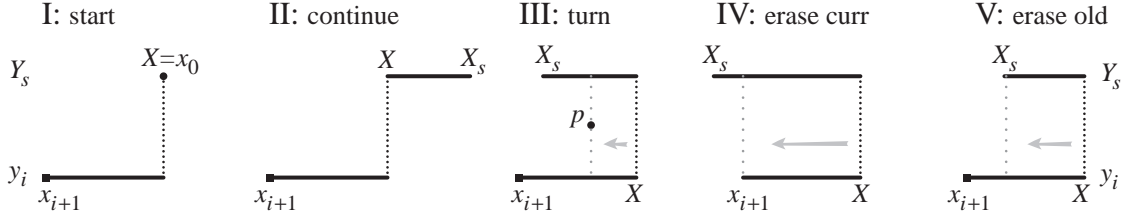


Figure 7: Five cases in incrementally computing a canonical path

Figure 7 illustrates cases (I)–(V) as we consider the current segment from (X, y_i) to (x_{i+1}, y_i) . We assume that $x_{i+1} < X$, since the reverse is symmetric.

If we are not doubling back—that is, if (I): $X = x_0$ or (II): $x_{i+1} < X < X_s$ —then push y_i and x_{i+1} , and increment i . The canonical sequence for this path will not gain any repeated pair of symbols.

If we are doubling back, $X_s \leq X$ and there are three cases. Either (III): there is a turn point p with $\max\{X_s, x_{i+1}\} < p.x \leq X$, or there is no turn point and (IV): $X_s < x_{i+1} < X$ or (V): $x_{i+1} < X_s \leq X$. We check for a turn point by performing an orthogonal range query with the rectangle $[\max\{X_s, x_{i+1}\}, X] \times [Y_s, y_i]$. We handle each case separately.

- (III): Turn point $(p.x, p.y)$ is found, so replace X with $p.x$ at the top of S , then push y_i and x_{i+1} , and increment i . Because of the turn point, adjacent symbols in the canonical sequence will not be identical. (Note: we may wind over and under p in this case.)
- (IV): No turn point; erase current segment. Replace X with x_{i+1} at the top of S and increment i . This shortens the path stored in S , so cannot create repeated symbols.
- (V): No turn point; erase old segment. Pop stack S twice to remove the old segment at the top. This case may apply repeatedly, as it does not increment i , but each repetition shortens the path in S .

We continue until $i = n$.

THEOREM 5. *The algorithm above correctly computes a path whose sequence is the canonical sequence for (α, P) using at most $2n$ orthogonal range queries plus $O(n)$ time.*

PROOF. First, we show that the algorithm terminates after $2n$ iterations. Each case (I)–(IV) increments the loop variable i and pushes at most two elements onto S . Case (V) pops S twice and leaves i unchanged. Initially $i = 0$, and we stop when $i = n$, so we have n iterations in total with cases (I)–(IV) and at most $2n$ elements pushed onto S . Case (V) never empties the stack—the sentinel x_0 triggers case (I)—so we have at most n iterations with case (V).

Using the arguments given in the description of the cases, we can inductively establish the invariant that, after i segments, stack S contains a canonical path starting from x_0 and ending with x_i . When the algorithm terminates at $i = n$, we have our canonical path. \square

3.5 Comparing canonical paths

Canonical sequences can have quadratic size, as the example of Figure 2 shows, so we still cannot compute the sequences for the canonical paths from (α, P) and (β, P) .

Fortunately, the canonical paths can be compared by a more complicated version of the sweep algorithm for rectifying paths.

Geometrically, the algorithm of the previous section contracts portions of the path α to vertical line segments. If we “unrectify” the canonical path contained in the stack S , we obtain a path $\hat{\alpha}$ that consists of fragments of α connected by vertical line segments. We can think of $\hat{\alpha}$ as a sequence of monotone chains delimited by turn points: let the path starting point be p_0 , the path end point be p_m , and the i^{th} turning point we encounter as we trace $\hat{\alpha}$ be p_i ; the i^{th} monotone chain α_i is defined as the portion of the path $\hat{\alpha}$ from p_{i-1} to p_i .

Using this notation, we first state two lemmas that will allow us to compare canonical paths, then we describe a sweep algorithm to perform the comparison. Finally, we show that the number of self-intersections is linear, which is necessary to make the sweep efficient.

LEMMA 6. *If the paths $\hat{\alpha}$ and $\hat{\beta}$ have the same canonical sequences, the corresponding monotone chains α_i and β_i are defined by the same two end points p_{i-1} and p_i .*

PROOF. Each turning point p_i corresponds to a label pair with opposite bars ($\overline{p_i} \underline{p_i}$ or $\underline{p_i} \overline{p_i}$) in the canonical sequence. Since the path α and β have the same canonical sequences, the sequence of point label pairs with opposite bars are also the same. Therefore, the sequence of turning points are the same. By the definition of monotone chain, each monotone chain α_i and β_i has the same end points. \square

Lemma 6 gives a necessary condition for $\hat{\alpha}$ and $\hat{\beta}$ to have the same canonical sequences: their monotone chains must have the same end points. If this is not satisfied, we can report that $\hat{\alpha}$ and $\hat{\beta}$ do not have the same canonical sequence, therefore α and β are not homotopic. Otherwise, further tests are needed.

LEMMA 7. *For each point p in the plane, let $A(\alpha, p) \subset \{1, 2, \dots, m\}$ be the set of indices of monotone chains $\{\alpha_i\}$ that lie above p . Similarly, define $A(\beta, p)$ as the set of indices of monotone chains $\{\beta_i\}$ that lie above p . If Lemma 6 is also satisfied, then the paths $\hat{\alpha}$ and $\hat{\beta}$ have the same canonical sequence if and only if $A(\alpha, p) = A(\beta, p)$ for each $p \in P$.*

PROOF. To compare the canonical sequences, we can traverse the path α' and β' with the same speed along the x -coordinate. Since the end points of each monotone chain α_i are the same as β_i , we pass the same sequence of points above or below in the plane as we traverse $\hat{\alpha}$ and $\hat{\beta}$. Suppose that monotone chains α_i and β_i are being traversed (since their indices are both i) and we pass a point p . If the canonical sequences for $\hat{\alpha}$ and $\hat{\beta}$ are the same, this point has the

same bar (\bar{p} or \underline{p}), so index i is either in both $A(\alpha, p)$ and $A(\beta, p)$ or in neither. Conversely, if $A(\alpha, p) = A(\beta, p)$, then p will have the same bar in the canonical sequences for $\hat{\alpha}$ and $\hat{\beta}$. \square

The result of Lemma 7 allows us to compare the canonical sequences using a sweep algorithm. The input of the algorithm is the set of α -monotone-chains and β -monotone-chains. We sweep these chains together from left to right with a vertical line. We maintain three invariants during the sweep:

1. We know the set of α -monotone-chains and the set of β -monotone-chains intersected by the sweep line at position x .
2. Within each set, we know the aboveness order of the monotone chains and the rank of each chain in this order. (It should be noted that the monotone chains could intersect, so this aboveness order depends on the current sweep line position x . We keep the orders for α and for β separate, otherwise we would need to compute all intersections between α and β .)
3. For each rank $1 < k < m$ and $\gamma \in \{\alpha, \beta\}$, we define *above set* $A(\gamma, k)$ as the set of indices of monotone chains from γ having rank at most k . We maintain the size of symmetric differences between $A(\alpha, k)$ and $A(\beta, k)$, and denote these *difference numbers* $D_x(k) = |\text{diff}(A(\alpha, k), A(\beta, k))|$.

The invariants allow us to check the hypothesis of Lemma 7 to compare the canonical sequences. When the sweep encounter a point p , we locate p in the set of ordered α -monotone-chains and the set of β -monotone-chains. This gives us the sizes of the above sets, $|A(\alpha, p)|$ and $|A(\beta, p)|$. If these sizes are not equal, then the canonical sequences are not the same according to Lemma 7. Otherwise, we let k equal this size, and check if $D_x(k) = 0$. If it is not, $A(\alpha, p) \neq A(\beta, p)$, and Lemma 7 again tells us that the canonical sequences are not the same. If it is, we continue. If these conditions are satisfied by all $p \in P$, then the canonical sequences are the same by Lemma 7.

Having shown that the invariants allow us to compare canonical sequences, we give algorithms to maintain these invariants and establish its complexity. To maintain the aboveness order and the difference numbers, we keep these data structures:

- Two balanced binary search trees that use aboveness as the order: T_α stores the α -monotone-chains indices, and T_β stores β -monotone-chain indices. Trees T_α and T_β allow us to search for the interval at which a point splits the monotone chains into above sets and below sets. By keeping counts in each subtree, we can also find the sizes of these sets.
- A balanced binary tree T_D that stores difference numbers $D_x(k)$ and uses k as the order. Difference numbers may be inserted or deleted.

Although we usually maintain these structures by handling events during a sweep, let us briefly describe how to initialize these data structures for a given sweep position x .

LEMMA 8. *The sweep data structures can be initialized for n chains at a sweep position x in $O(n \log n)$ time.*

PROOF. We need to form T_α , T_β , and T_D , as defined above. For $\gamma \in \{\alpha, \beta\}$, order the chains of γ by aboveness to form T_γ .

To form T_D , we must compute the difference numbers $D_x(k)$, for $k = 1 \dots m$. First build two auxillary tables: Let $I_\gamma(r)$ be the chain index at a given rank in T_γ and $R_\gamma(i)$ be the rank of a given index; these are inverses $I_\gamma(R_\gamma(i)) = i$ and $R_\gamma(I_\gamma(r)) = r$. Initialize $D_x(k) = 0$. Then, for $k = 1 \dots m$, compute

$$D_x(k) = D_x(k-1) + \text{sgn}(R_\beta(I_\alpha(k)) - k) - \text{sgn}(R_\alpha(I_\beta(k)) - k).$$

This simply says that the difference number increases by one whenever the k th chain in the α list ranks higher in the β list and decreases when it ranks lower, with the reverse for the k th chain in the β list.

The most time-consuming step of this computation is sorting and ranking, which takes $O(n \log n)$ time for n chains. \square

We maintain the data structure as we sweep the plane with a vertical line. At certain *events*, the sweep reaches the x -coordinate of a point at which we must update the data structures or compare the canonical sequences. In order to do this, the events clearly need to include monotone chain end points and points in the plane. Since the monotone chains from one path could intersect each other, the events also need to include all self-intersections of $\hat{\alpha}$ and $\hat{\beta}$. These intersections are computed during the sweep [3].

Before we describe how each event is handled, we make some simplifying assumptions about the monotone chains so that the algorithm is easier to describe—we will remove these assumptions later. First, we will treat each monotone chain as strictly monotone, even though chains from the canonical path may contain vertical line segments. Second, we will assume that no two event points have the same x -coordinate, even though this is not the case because of the vertical line segments. Now, we consider five types of events.

- (I) point in the plane: We check that the canonical sequences are still the same. First, locate the point among the monotone chains using T_α and T_β , then check that the difference number is zero, thereby verify that the α above set is the same as the β above set.
- (II) intersection event: Suppose the intersection is caused by monotone chains α_i and α_j so that α_i is below α_j after the intersection. To update T_α and T_β , we simply swap the monotone chain indices i and j . To update the T_D , first find the aboveness rank k of α_i . Then, we know that the monotone chain α_i has left the aboveness set $A(\alpha, k)$; and the monotone chain α_j has entered $A(\alpha, k)$. Recall that the difference number $D_x(k) = |\text{diff}(A(\alpha, k), A(\beta, k))|$. So it is clear that $D_x(k)$ —and only $D_x(k)$ in T_D —needs to be changed. To change $D_x(k)$ appropriately, we find out whether the monotone chain β_i is in $A(\beta, k)$. If it is, then increment $D_x(k)$. Otherwise, do nothing. Symmetrically, if the monotone chain β_j is in $A(\beta, k)$, decrement $D_x(k)$. Otherwise, do nothing.
- (III) path end point: At a path endpoint we must have either both the α chain and β chain begin or both end

for the paths to be homotopic. Since there are only two path endpoints, we use Lemma 8 to reinitialize the data structures at these events.

- (IV) chain starting event: At a turn point, we may have m monotone chains begin in α and in β . We first apply Lemma 8 just to the m new chains to compute the difference numbers restricted only to the new chains, which we denote $D'_x(r)$, for $r = 1 \dots m$.

To update the algorithm's data structures, we next locate the event point in the trees T_α and T_β and then insert the new chains in these trees in the aboveness order.

We must now update the difference numbers in T_D . Note that because the event is a turning point, if the insertion does not take place at the same rank k in both T_α and T_β , or if different numbers of α and β chains are inserted, then the paths are not homotopic by Lemma 7. Thus, we need only insert m new differences; the old difference numbers do not otherwise change. Since the r^{th} interval in the new monotone chains becomes the $(r+k)^{\text{th}}$ interval after the insertion, we insert the new difference numbers $\{D_x(r+k) = D'_x(r) + D_x(k), r = 1, 2, \dots, m\}$ into T_D .

- (V) chain ending event: The operations for updating data structures at a turning point where m monotone chains end are exactly the reverse of the operations at a chain starting event. To update T_α and T_β , we delete the indices of all monotone chains ending at the event point. Assuming that these had ranks $k+1$ through $k+m$, we delete the difference numbers $\{D_x(k+1), D_x(k+2), \dots, D_x(k+m)\}$ from T_D .

The number of intersection events depends on the definition of path that we use, but we will show a linear upper bound in the next section. Thus, our sweep to compare canonical sequences is efficient.

THEOREM 9. *Linear space and $O(n \log n)$ time are sufficient to compare the canonical sequences of canonical paths $\hat{\alpha}$ and $\hat{\beta}$, assuming that the number of self-intersections in each canonical path is $O(n)$.*

PROOF. The space is linear since all the data structures we used are linear (T_α , T_β and T_D). To see the time, note that each operation to maintain T_α and T_β (insertion or deletion of the chain indices) can be charged to a monotone chain end point, and each operation for maintaining T_D (computing, insertion or deletion of the difference numbers) can be charged either to a monotone chain end point or an intersection point. Since the time for each of these operations is at most $O(\log n)$, and we assume that there are $O(n)$ number of monotone chain end points and intersection points, the total time is $O(n \log n)$. \square

By careful implementation, we can get rid of the assumptions that simplified the algorithm description. First, we assumed that the chains were strictly monotone, but in fact they contain vertical line segments. Therefore, in order for intersection points to be handled correctly, the sweep line should stop at a vertical line segment, compute the intersections on it and order these intersection events by the order they are encountered by the path. Second, we assumed that no two events have the same x-coordinate, although some

points in the plane, such as the turning points, lie on vertical line segments. To handle these points correctly, we need to perturb them away from the vertical line segments in the right direction: from the canonical path computation, we know whether the line segment is some path contracted from the right side of the point or from the left. If it is from the left, this event should be handled immediately after the vertical line segment; if it is from the right, it should be handled immediately before the vertical line segment.

3.6 Modifications for path definitions

We have described the algorithm for the standard (tack) definition of path homotopy. For the pin and pushpin definitions, we need only a few modifications.

To compute a canonical sequence, we add path endpoints to the set of obstacle points P . Under the pushpin definition, we add two points that are infinitesimally left and right of the start point. Then the algorithm of Section 3.4 correctly winds around the start in the pushpin case, and unwinds in the pin case. For the pin case we may need to unwind around the end point in postprocessing.

As we show in the next two lemmas, a canonical path under the pin and pushpin definitions has no self-intersections, whereas a canonical path under the tack definition has a linear number of self-intersections. The sweep algorithm to compare canonical paths (Section 3.4) can be greatly simplified for pins and pushpins. All sorting can be performed at the beginning using the aboveness ordering, and the intersection case (II) never occurs. We need not even maintain the table of difference numbers, since they change only on insertion and deletion of a chain.

LEMMA 10. *Under the pin and pushpin definitions, a simple path α has a canonical path $\hat{\alpha}$ that is also simple.*

PROOF. Because the path α is simple, intersections occur only if some vertical line segment $L \subset \hat{\alpha}$, coming from the contraction of a portion $\theta \subset \alpha$, intersects some point $q \in \hat{\alpha} \cap \alpha$. Let γ be the portion of $\alpha \setminus \theta$ that contains q —note that one end of γ is an endpoint of α , and the other end is open with limit point at an endpoint of both L and θ . We will show that the existence of q contradicts the assumption that $\hat{\alpha}$ is a canonical path.

Lift L and θ to L' and θ' in the universal cover. The pair (L', θ') defines a region R' in the universal cover, which must necessarily be simply connected. Lift γ to γ' containing $q' \in L'$. Since θ and γ are portions of a curve with no self-intersections, γ' cannot cross θ' . If we trace γ' into R' , no matter whether we trace toward the endpoint of α or toward the endpoint of θ' , then we must encounter another intersection with L' : in the first case, because R' cannot contain a path endpoint under the pin or pushpin definition, and in the second, because the endpoint of θ' is itself on L' . But Corollary 2 implies that this portion of γ would be contracted to an empty sequence, eliminating the intersection at q in the canonical path $\hat{\alpha}$. This contradiction establishes the lemma. \square

We can limit the number of self-intersections of a canonical path under the tack definition.

LEMMA 11. *A n -segment simple path α under the tack definition has a canonical path $\hat{\alpha}$ with $\leq 2n$ self-intersections.*

PROOF. As in the previous lemma, an intersection occurs only if some $\theta \subset \alpha$ was contracted to a vertical line segment

$L \subset \hat{\alpha}$ that intersected $\hat{\alpha}$. Figure 8(a) shows an example having two intersections with L .

To find the number of intersections on a vertical segment L , we examine the original path before and after θ in the universal cover. Define the size $|\theta|$ as the number of vertices in the chain θ , not counting those that are the endpoints of L . Let $\gamma \subset \alpha$ be the path after θ , and $\hat{\gamma} \subset \hat{\alpha}$ be the canonical path after L , which is homotopic to γ .

We claim that $\hat{\gamma}$ intersects L at most $|\theta|$ times, which will imply that L has at most $2|\theta|$ intersections. Because the vertical segment of the canonical path arise by contracting disjoint portions of the original path α , the total number of intersections is at most $2|\alpha| = 2n$. Thus, it is sufficient to establish this claim.

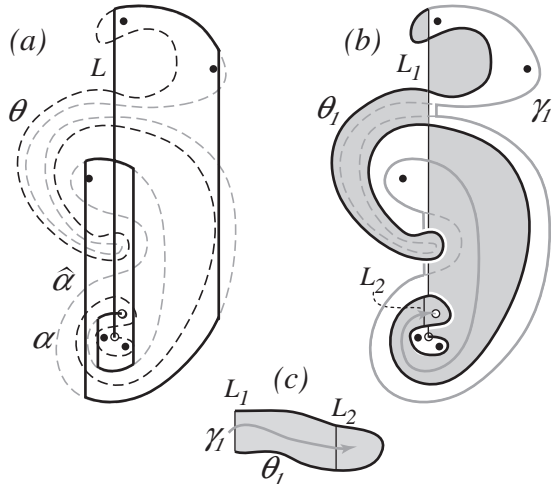


Figure 8: In (a), curve α (dashed) has a self-intersecting canonical path $\hat{\alpha}$ (solid), in which θ (dark dashes) contracts to vertical segment L . In (b), we lift to the universal cover and trace γ to find intersected copies of L , which in (c) are depicted schematically.

Let $\{L_i\}$ denote the set of “copies” of L created by lifting L to the universal cover. By applying Corollary 2, we can find a path γ_1 in the universal cover that is homotopic to γ and does not intersect a copy more than once. Thus, we can number the copies $L_1 \dots L_k$ in the order that γ_1 intersects them. Figure 8 (b) shows γ_1 intersecting L_1 and L_2 .

The pair (L_1, θ_1) , where θ_1 is the lift of θ starting at the endpoints of L_1 , bounds a region R' in the universal cover that contains γ_1 and all its intersections. Figure 8 (c) shows this schematically. If we follow θ_1 around the boundary of this region, then we encounter a vertex of θ_1 to the right or left of L between each endpoint of a copy of L . Each vertex is encountered at most twice; once from each side of θ_1 . Therefore, if there are k intersections, we have $2|\theta| \geq 2k - 1$, or $k \leq |\theta| + 1/2$. Since k is an integer, $k \leq |\theta|$. \square

4. HOMOTOPY FOR NON-SIMPLE PATHS

When paths are allowed to intersect themselves, as well as each other, then we cannot use aboveness to rectify the paths as in the previous section. In this section we show that we can use a different construction of a universal cover and achieve a running time near $O(n^{3/2})$.

Since the paths may self-intersect, we simply concatenate them as suggested in the introduction and consider whether a closed loop α is contractible in the plane minus P . The idea is to lift the path into a universal cover constructed in an appropriate way, and test if the endpoints of the lifted path coincide. We will use a simple path d going through all points in P in order to construct such a universal cover, but, in order to minimize the number of operations during the lifting of α , we have to minimize the number of intersections between d and α . The way to construct d is using a spanning tree with low crossing number.

LEMMA 12. *A simple spanning tree of P with crossing number $O(\sqrt{n})$ can be constructed in $O(n^{1+\epsilon})$ time, for any $\epsilon > 0$.*

PROOF. We combine ideas and results from [12, 15, 19, 20]. Start by computing a spanning tree T_1 of P with crossing number $O(\sqrt{n})$. In [20] it is shown how a simplicial partition $\{P_1, \dots, P_{n/2}\}$ with $|P_i| \leq 4$ and crossing number $O(\sqrt{n})$ can be constructed in time $O(n^{1+\epsilon})$. This partition plays the role of the partial matching used in [19]: for each class P_i with $|P_i| \geq 2$, we put an edge between a pair of points lying in it, and remove anyone of its endpoints from the set of points P . Iterating this step $O(\log n)$ times, we get the spanning tree T_1 within $O(n^{1+\epsilon})$ operations.

Tree T_1 can have self-intersections, but we can use it to construct a Steiner spanning tree T_2 that is simple [12]. We start with an empty T_2 . Then, we traverse T_1 in preorder, and when we visit a vertex $p \in P$, we shoot a ray along the edge connecting p toward its parent. Then we add to T_2 the segment from p to the first intersection of the ray with current T_2 . The dynamic data structure for ray shooting described in [15] supports updates and queries in $O(\log^2 n)$ time, so T_2 can be constructed in $O(n \log^2 n)$ time. Observe that T_2 has $O(n)$ edges, and that, as subset of T_1 , it also has crossing number $O(\sqrt{n})$.

As proved in [12], from T_2 we can construct in $O(n \log n)$ time a spanning tree T_3 with no Steiner points, and whose crossing number is twice that of T_2 . \square

THEOREM 13. *We can decide if a closed loop α is contractible in the plane minus P in $O(n^{3/2} \log n)$ time.*

PROOF. Take a bounding box B enclosing α and P , and let p_- and p_+ be two points on the left and on the right of B , respectively. We apply the previous lemma to the set $P \cup \{p_-, p_+\}$ to obtain a simple spanning tree, and then we use it to make a tour starting at p_- and finishing at p_+ . If we perturb some edges of this tour slightly, we can convert it into a simple path d from p_- to p_+ , passing through all points, that has crossing number $O(\sqrt{n})$.

This path d splits B into two pieces, B_0 and B_1 , which we can use as building blocks for the portion of a universal cover that contains the lift of the loop α . We actually build the dual graph T (a tree) for the universal cover, and keep just one copy of each block. Thus, we can represent the portion of the universal cover containing the lift of α in space proportional to n plus the number of nodes of the dual tree T .

Here is the construction of T . Preprocess blocks B_0 and B_1 for ray shooting [17]. Start with some vertex of α in B_i , where bit $i \in \{0, 1\}$, and create a tree node ν to represent this copy of B_i .

Now, use ray shooting along the segments of α to trace α in B_i until it leaves by crossing an edge e of the path d . If edge e has never been crossed before, then make a new node η in the dual tree T , put an edge between current node ν and η , annotated with the edge e . If edge e has been crossed, then ν has an edge annotated with e that connects to a node η . In either case, update the current node $\nu = \eta$, and complement i to switch to the other block. When we complete the tracing, we know that α is contractible if and only if we ended up in the same node of T in which we started.

Because each edge of T comes from one intersection of d and α , and given that the crossing number of d is $O(\sqrt{n})$, tree T has size $O(n^{3/2})$. An additional logarithmic factor in the running time is sufficient to pay for the ray shooting. \square

5. LOWER BOUNDS

In this section we establish lower bounds for the path homotopy question among points in the plane. The bounds are different if the paths are simple or not.

5.1 Simple paths

When both paths α and β are simple, to decide if they are homotopic takes a minimum of $\Omega(n \log n)$ operations in the decision tree model. In order to show this, consider the following problem: given a set of n disordered numbers x_1, \dots, x_n , and a set of n disjoint intervals $[a_1, b_1], \dots, [a_n, b_n]$, ordered by increasing order, is there any interval containing any point? Using Ben-Or's theorem [2], it is straightforward to see that in the algebraic decision tree model, this problem has a lower bound of $\Omega(n \log n)$.

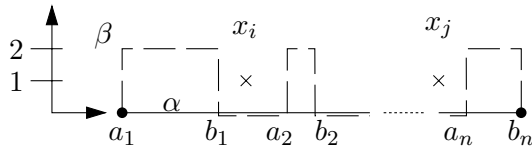


Figure 9: Lower bound for simple paths

But this problem can be reduced to a test whether two simple paths are homotopic. Consider α to be the segment with endpoints $(a_1, 0)$ and $(b_n, 0)$, and β to be the path following α except when it overlaps with some interval $[a_i, b_i]$, in which case it will follow the horizontal segment at height 2, as shown in Figure 9. If we consider the set of points $P = \{(x_i, 1) \mid 1 \leq i \leq n\}$, then no point belongs to any interval if and only if the paths α and β are homotopic.

5.2 Non-simple paths

When the polygonal lines α and β are allowed to self-intersect, then the problem becomes harder: we can reduce Hopcroft's problem to the problem of testing if two paths are homotopic.

Given n points and n lines, *Hopcroft's problem* asks if any point lies on any line. The best algorithms known to solve Hopcroft's problem take just slightly more than $O(n^{4/3})$ time [6]. Erickson showed that *partition algorithms*, a certain class of algorithms that includes the natural and known algorithms, take at least $\Omega(n^{4/3})$ time [14].

We give a reduction that works for partition algorithms, and any other algorithm whose primitive tests check the

signs of bounded-degree polynomials of the input point coordinates. We need this restriction so that we can use infinitesimal quantities in the construction. As is common in perturbation methods, we can expand the primitives as polynomials in one infinitesimal variable and determine the sign of the term of smallest degree [13].

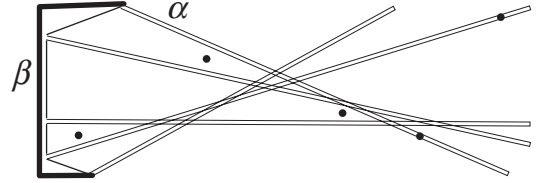


Figure 10: Reduction from Hopcroft's problem

Given an instance of Hopcroft's problem, determine a bounding box that contains all the points and intersects the faces of the arrangement of lines that are above all lines and below all lines. Let p and q be points on the bounding box that lie in these faces and, as illustrated in Figure 10, this can be done in linear time after sorting the lines by slope. Let β be the left path along the bounding box from p to q . Let α be the path that follows β except for leaving the bounding box at every line, crossing the box infinitesimally above the line, and returning infinitesimally below the line to β . Paths α and β are homotopic iff no point lies on any line.

6. CONCLUSION

We have given an efficient, elementary algorithm to test homotopy for simple paths in the plane by using an aboveness ordering to rectify paths. We believe that it may be possible to eliminate the range queries by adding segments to the path in aboveness order instead of traversal order, but each way we have tried so far slips back to a quadratic worst case. For non-simple paths, we use standard machinery to give a subquadratic algorithm. Lower bounds show that our first algorithm is tight, but that it may be possible to improve the second.

The first author is grateful to Mark de Berg and Marc van Kreveld for helpful discussions regarding this research. He also wants to thank Pankaj Agarwal for pointing him to hierarchical cuttings to reduce the construction time in Lemma 12.

7. REFERENCES

- [1] M. A. Armstrong. *Basic Topology*. McGraw-Hill, London, UK, 1979.
- [2] M. Ben-Or. Lower bounds for algebraic computation trees. In *Proc. 15th Annu. ACM Sympos. Theory Comput.*, pages 80–86, 1983.
- [3] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, C-28(9):643–647, September 1979.
- [4] B. Buttenfield. Treatment of the cartographic line. *Cartographica*, 22:1–26, 1985.
- [5] Bernard Chazelle. An algorithm for segment-dragging and its implementation. *Algorithmica*, 3:205–221, 1988.

- [6] M. de Berg and O. Schwarzkopf. Cuttings and applications. *Internat. J. Comput. Geom. Appl.*, 5:343–355, 1995.
- [7] M. de Berg, M. van Kreveld, and S. Schirra. A new approach to subdivision simplification. In *Proc. 12th Internat. Sympos. Comput.-Assist. Cartog.*, pages 79–88, 1995.
- [8] T. K. Dey and S. Guha. Transforming curves on surfaces. *Journal of Computer and System Sciences*, 58:297–325, 1999.
- [9] T. K. Dey and H. Schipper. A new technique to compute polygonal schema for 2-manifolds with application to null-homotopy detection. *Discrete Comput. Geom.*, 14:93–110, 1995.
- [10] J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. E. Tarjan. Making data structures persistent. *J. Comput. Syst. Sci.*, 38:86–124, 1989.
- [11] H. Edelsbrunner. A note on dynamic range searching. *Bull. EATCS*, 15:34–40, 1981.
- [12] H. Edelsbrunner, Leonidas J. Guibas, J. Hershberger, R. Seidel, Micha Sharir, J. Snoeyink, and Emo Welzl. Implicitly representing arrangements of lines or segments. *Discrete Comput. Geom.*, 4:433–466, 1989.
- [13] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.*, 9(1):66–104, 1990.
- [14] Jeff Erickson. New lower bounds for Hopcroft’s problem. *Discrete Comput. Geom.*, 16:389–418, 1996.
- [15] M. T. Goodrich and R. Tamassia. Dynamic ray shooting and shortest paths in planar subdivisions via balanced geodesic triangulations. *J. Algorithms*, 23:51–73, 1997.
- [16] J. Hershberger and J. Snoeyink. Computing minimum length paths of a given homotopy class. *Comput. Geom. Theory Appl.*, 4:63–98, 1994.
- [17] J. Hershberger and Subhash Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *J. Algorithms*, 18:403–431, 1995.
- [18] F. M. Maley. *Single-Layer Wire Routing and Compaction*. MIT Press, Cambridge, MA, 1990.
- [19] J. Matoušek. More on cutting arrangements and spanning trees with low crossing number. Technical Report B-90-2, Fachbereich Mathematik, Freie Universität Berlin, Berlin, 1990.
- [20] J. Matoušek. Efficient partition trees. *Discrete Comput. Geom.*, 8:315–334, 1992.
- [21] J. R. Munkres. *Topology: A first course*. Prentice Hall, Englewood Cliffs, NJ, 1975.
- [22] L. Palazzi and J. Snoeyink. Counting and reporting red/blue segment intersections. *CVGIP: Graph. Models Image Process.*, 56(4):304–311, 1994.
- [23] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 3rd edition, October 1990.
- [24] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.