Discrete Comput Geom 31:61–81 (2004)
DOI: 10.1007/s00454-003-2949-y

Discrete & Computational
Geometry

Testing Homotopy for Paths in the Plane*

Sergio Cabello,[1] Yuanxin Liu,[2] Andrea Mantler,[2] and Jack Snoeyink[2]

[1]Institute of Information and Computer Science, Utrecht University,
3508 TB Utrecht, The Netherlands
sergio@cs.uu.nl

[2]Department of Computer Science, University of North Carolina,
Chapel Hill, NC 27599-3175, USA
{liuy,mantler,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 paths specified by n line segments with obstacles described by n points, several standard ways achieve quadratic running time. For simple paths, 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; our algorithm runs in $O(n^{3/2} \log n)$ time.

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 (Fig. 1). Equivalently, we determine whether the closed loop $\alpha\beta^{\mathrm{R}}$, 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 collinear and no two points are on the same vertical line. We are primarily concerned with paths that have no self-intersections.

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

* The first author was partially supported by the Cornenlis Lely Stichting, and the rest by NSF Grants 9988742 and 0076984.

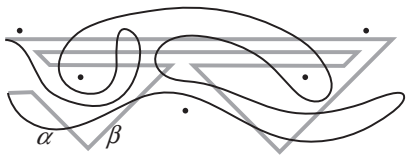

Fig. 1. Are α and β homotopic?

polynomial-time solvable variations of the wire routing problem [21], [17]. In motion path planning, one might check to see that two ways of getting from point A to point B are equivalent [19]. 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 [5], [8]. 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 60 m 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. Efrat et al. [15] recently looked at computing shortest paths among obstacles by identifying and bundling homotopic fragments. They independently developed a sweep algorithm that uses similar ideas to ours. Our rectification can be used to speed up their algorithm, which is further improved by Bespamyatnikh [4].

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 [19] trace each path through a triangulation, taking time proportional to the number of 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 Fig. 2. The more general problem of testing if two paths are homotopic in compact surfaces, with or without boundary, was treated in [9] and [10]. 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 nonsimple paths in Section 4, and establish lower bounds for both problems in Section 5. We conclude in Section 6.

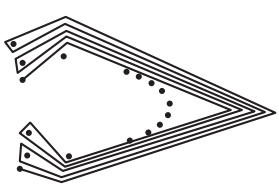

Fig. 2. $\Theta(n^2)$ segments in the shortest homotopic path.

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], [24]. Let $I = [0, 1]$ denote the unit interval and let 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 $\alpha: I \to M$; 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 $\Gamma: [0, 1] \times [0, 1] \to M$ such that $\Gamma(x, 0) = \alpha(x)$ and $\Gamma(x, 1) = \beta(x)$, and $\Gamma(0, y) = \alpha(0) = \beta(0)$ and $\Gamma(1, y) = \alpha(1) = \beta(1)$.

By the standard topological definition, path endpoints 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. If we fix the ends of the thread with tacks, pins, or pushpins, we can obtain three variations of the homotopy problem, as defined below:

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

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

pushpin A pushpin serves as a larger obstacle, so that closed ε neighborhoods around the the endpoints 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 Fig. 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 must pass over endpoints, temporarily creating a path with self-intersections. Paths γ and δ are homotopic under the tack and pin definitions, but are not homotopic under the pushpin definition, which preserves how γ winds around the left endpoint.

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 causes extra complications in handling self-intersections, especially in Section 3.6. (The work of Efrat et al. [15] uses the pushpin definition.)

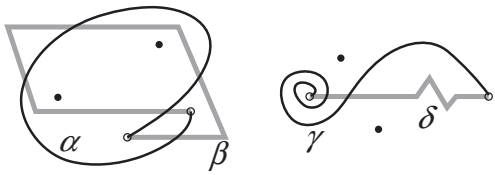

Fig. 3. Distinguishing between path definitions.

2.2. *Canonical Sequences*

Whatever definition of path we choose, path homotopy is an equivalence relation [1], [24], which implies we can identify a homotopy class by giving a representative path.

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 $\alpha = \underline{\mathrm{ABB}}\overline{\mathrm{BCD}}\underline{\mathrm{D}}\overline{\mathrm{CC}}\underline{\mathrm{DEED}}\overline{\mathrm{C}}\underline{\mathrm{BA}}$ and $\beta = \underline{\mathrm{A}}\overline{\mathrm{BCD}}\underline{\mathrm{EED}}\overline{\mathrm{DDCBB}}\underline{\mathrm{BA}}$. 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, $\underline{\mathrm{B}}\overline{\mathrm{B}}$ and $\overline{\mathrm{D}}\underline{\mathrm{D}}$. 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 $\underline{\mathrm{BB}}$ and $\underline{\mathrm{D}}\overline{\mathrm{CC}}\underline{\mathrm{DEE}}$, and from β we delete $\underline{\mathrm{EE}}$, $\overline{\mathrm{DD}}$, and $\overline{\mathrm{BB}}$, to find that both have the same canonical sequence $\underline{\mathrm{A}}\overline{\mathrm{BCD}}\underline{\mathrm{D}}\overline{\mathrm{C}}\underline{\mathrm{BA}}$. 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*

The topological concept of covering space has been used in the works of Gao et al. [17] and Hershberger and Snoeyink [19]. We describe briefly its properties here; please refer to basic topology texts [1] and [24] for a more through description.

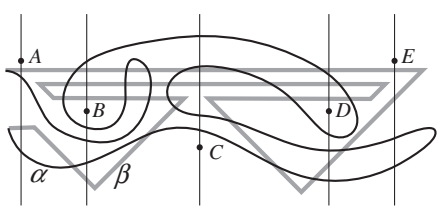

Fig. 4. Paths $\alpha = \underline{\mathrm{ABB}}\overline{\mathrm{BCD}}\underline{\mathrm{D}}\overline{\mathrm{CC}}\underline{\mathrm{DEED}}\overline{\mathrm{C}}\underline{\mathrm{BA}}$ and $\beta = \underline{\mathrm{A}}\overline{\mathrm{BCD}}\underline{\mathrm{EED}}\overline{\mathrm{DDCBB}}\underline{\mathrm{BA}}$ have the same canonical sequence $\underline{A}\overline{BCD}\underline{D}\overline{C}\underline{BA}$.

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], [24]. 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 (Fig. 4), this space is most easily described by a 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 nonempty, 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], [24]. 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 endpoint. 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. □

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. □

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 [26]: $A \succ B$ 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 [1], [24] 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, we can easily show that the aboveness relation is a partial order [3]:

Lemma 3. *For pairwise disjoint, vertically convex sets $A_1, A_2, \ldots, A_k$ 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 [25].

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 this k-ary tree as a binary tree by giving each path a pointer to its left sibling and rightmost child, as in Fig. 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

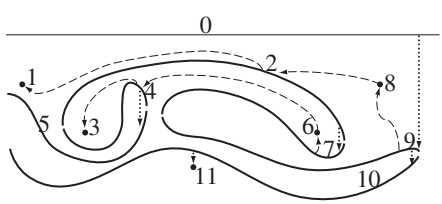

Fig. 5. Inorder numbering of the child/sibling representation of the aboveness tree for path α.

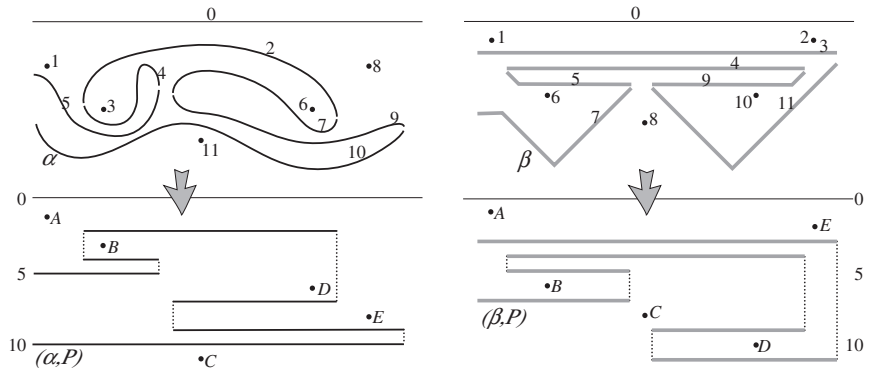

Fig. 6. Numberings for α and β are used to form rectified pairs (α, P) and (β, P). Rectifying preserves path simplicity and sequences.

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 [25] if more detail is desired.

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 [6]. The RT of Edelsbrunner [12] or the range priority tree of Samet [27] 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 [6]. We describe a simple structure with $O(n \log n)$ space based on persistence [11] in the hope that it can be improved: Sort the points by increasing x coordinate, so that $p_1, p_2, \ldots, p_n$ are ordered from left to right. Denote the prefixes by $P_i = \{p_1, p_2, \ldots, p_i\}$, 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. □

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. We call this new path a *rectified canonical path*. It must be understood that this is defined with respect to a rectified pair, and not the original points. At the end of this subsection, we derive nonrectified canonical paths for α, β, and the original point set P.

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

The canonical path that we compute is determined from (α, P) by an important *last intersection property*. Given any oriented curve γ in the plane, we say that a curve $\hat{\gamma}$ from the homotopy of γ has the last intersection property if any vertical segment L in the universal cover intersecting the lift of $\hat{\gamma}$ does so at the last of its intersections with the lift of γ. In Fig. 7 the dark curve $\hat{\alpha}$ has the last intersection property in the universal cover. For example, segment L_1 has three intersections with the lift of α, and the last is in the lift of $\hat{\alpha}$. Segment L_2 actually has only one intersection with the lift of α, which is in $\hat{\alpha}$.

We consider the horizontal segments of α in order. We maintain the invariants that after i segments, for $0 \leq i \leq n$:

1. Stack S contains a rectified canonical path for the prefix of α from x_0 to x_i.

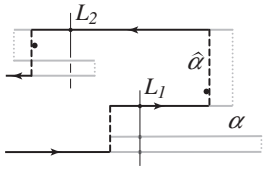

Fig. 7. $\hat{\alpha}$ has the last intersection property.

2. The canonical sequence for the path on S has no adjacent repeated symbols.
3. The path on S has the last intersection property.

Begin by setting $i = 0$ and pushing x_0 onto S. This establishes the initial invariant for the stack; the others are trivially true.

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 on the stack. Figure 8 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 and the path is extended in the current direction.

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, and vertical lines in the universal cover will intersect a single horizontal segment. (Note: we may wind over and under p in this case; sometimes it helps to think of p as a pair of points separated infinitesimally in x so that winding generates infinitesimal horizontal segments.)

(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 or violate the last intersection property.

(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.

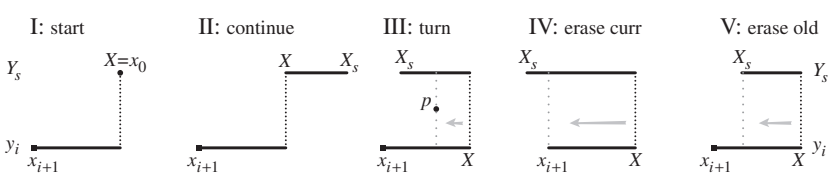

Fig. 8. Five cases in incrementally computing a canonical path.

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 establish and inductively maintain the invariants that, after i segments, stack S contains a rectified canonical path, with the last intersection property, that starts from x_0 and ends with x_i. When the algorithm terminates at $i = n$, we have our rectified canonical path. □

Once we have run the algorithm, we can unrectify the path, as in Fig. 9, to return to the original set of points. This will allow us to compare two paths on the same point obstacles. The rectified canonical path uses horizontal segments, corresponding to segments of α, and vertical segments corresponding to portions that have been shrunk by homotopy. Since the x-coordinates of all points and segments were preserved in rectification, and the aboveness relation was respected, we can easily map back to the original portions of α and join them by vertical segments, forming canonical path $\hat{\alpha}$. The last intersection property (recall Fig. 7) is also preserved.

We have described the algorithm for the standard (tack) definition of path homotopy. For the pin and pushpin definitions, we add path endpoints to the set of obstacle points P. Under the pushpin definition, we also add two points that are infinitesimally left and right of the start point to make the algorithm wind correctly around the start. Under the pin definition, the algorithm unwinds at the start, and we postprocess, if needed, to unwind at the endpoint.

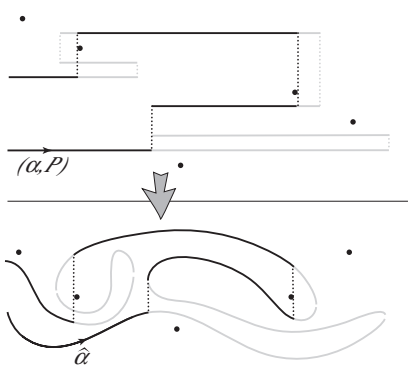

Fig. 9. Unrectifying the canonical path for (α, P).

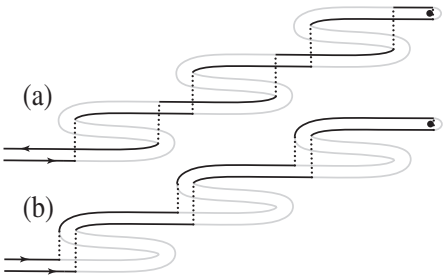

Fig. 10. Self-intersections in (a) avoided by leftist path (b).

3.5. *A Leftist Path*

Figure 10(a), adapted from [15], illustrates that a canonical path with the last intersection property need not be simple. In fact, one can create examples with a quadratic number of self-intersections. We define the *leftist canonical path* $\hat{\alpha}$ and prove that under the pin and pushpin definitions it has no self-intersections and under the tack definition it has $O(n)$ intersections.

The path is called "leftist" because it consists of monotone fragments that are traversed from left to right, so the vertical segments chosen for homotopy are as far to the left as possible, as in Fig. 10(b). The idea is simple: put together fragments of two canonical paths to change the last intersection property to a *left intersection property*: any vertical segment in the universal cover either does not intersect the lift of the path, or intersects it in the point where the lift of α last crosses from left to right.

For a rectified pair (α, P), let α^{F} and α^{R} (forward and reverse) be the two canonical paths obtained by running the algorithm of the previous section twice, starting from either end of the curve α. Since both paths are homotopic to α, the canonical sequence of α^{R} is the reverse of that of α^{F}—both have the same turn points, which we can number p_1 to p_{m-1} along α^{F}. (Some points may be duplicated, and we use infinitesimal horizontal segments for portions that wind around a point.) Let p_0 denote the start point and p_m the endpoint.

We snip α^{F} and α^{R} at the turn points. Fragments ending below a turn point are extended infinitesimally beyond the turn point by an amount proportional to their rank order below the turn point. For example, if there are three fragments whose right endpoints are below a turnpoint $p \in P$, then the lowest one is extended 3ε to the right, the next 2ε, and the highest by ε. To assemble these fragments into the *leftist path* $\hat{\alpha}$, we start at point p_0, with $i = 1$. We take the fragment α_i from α^{F} if p_{i-1} is left of p_i, otherwise we take the fragment α_i from α^{R}. We connect fragments in order, extending fragments above turn points to match the extensions below.

Notice that $\hat{\alpha}$ has the same canonical sequence as α—since it was derived from fragments of α^{F} and α^{R}, it has the same turn points and the same sets of points above and below. Thus, $\hat{\alpha}$ is homotopic to α.

A vertical segment within a fragment $\alpha_i \subset \hat{\alpha}$ has the left intersection property, because it had the last intersection property with either α^{F} or α^{R}, whichever came from the left. At the left endpoints, fragments have the left intersection property for any vertical line

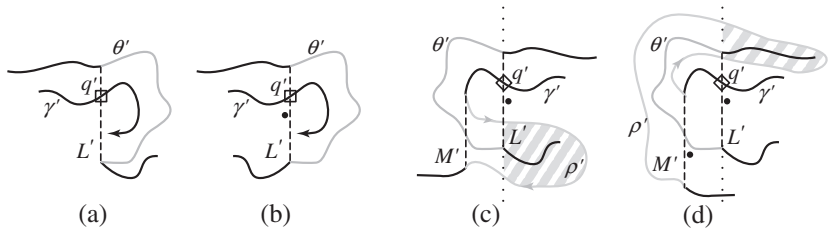

Fig. 11. Examples lifting γ' starting from q' within $R' = R(L', \theta')$. (Striped areas are used toward the end of the proof of Lemma 6.)

just to the right of the obstacle point; at the right endpoints, they have the left intersection property for vertical lines just to the left of the obstacle point. We use these properties as we consider self-intersections of the leftist path $\hat{\alpha}$ in the next lemma.

Lemma 6. *Consider the leftist canonical path $\hat{\alpha}$ for a simple path α with n segments: Under the pin and pushpin definitions $\hat{\alpha}$ is also simple. Under the tack definition $\hat{\alpha}$ has at most $2n$ points of self-intersection.*

Proof. On the leftist path $\hat{\alpha}$, consider any vertical segment $L \subset \hat{\alpha}$, from the homotopy of a portion $\theta \subset \alpha$. Let p be the start point of curve α, and let γ be the connected portion of $\alpha \backslash \theta$ that contains p. We lift γ to the universal cover to consider whether a point q where γ intersects L can be a self-intersection of $\hat{\alpha}$.

Lift L and θ from either of their shared endpoints to obtain L' and θ', which bound a simply connected region $R' = R(L', \theta')$ in the universal cover. Lift intersection point q to $q' \in L'$, and lift curve γ to γ' starting from point q'. If this lifts p to a point p' that lies in R', then we say that q is a *normal* intersection, otherwise q is *special*. Point q is normal in Fig. 11(a),(b) and special in (c),(d). We aim to count self-intersections of $\hat{\alpha}$ by separately counting the normal and special intersections that can create them. Figure 12 shows a complex example (under the tack definition) in which $L \cap \gamma$ has both normal and special intersections that appear as self-intersections of $\hat{\alpha}$.

Normal intersections actually come in equivalence classes: a given lift of p to $p' \in R'$ can contribute several intersections $L' \cap \gamma'$, all of which will be normal. We consider all of these to be in the same equivalence class, by Corollary 2. In fact, each class can contribute at most one self-intersection of $\hat{\alpha}$ by the left intersection property. We therefore count the number of normal intersection classes.

Under the pin and pushpin definition there are no normal intersection classes—endpoint p is an obstacle, so p cannot be lifted to lie inside the simply connected region R'.

Under the tack definition the number of normal intersection classes for L and γ is the number of copies of point p that can be lifted into R'. (Two in Fig. 12, for example.) To count copies, from each copy p' in the universal cover draw a vertical ray downward to hit a segment of θ'; each segment of θ' (except the one hitting the top of L') can be hit by at most one ray in the universal cover, since all the vertical rays have the same projection into the plane. Thus, R' contains at most $|\theta|$ distinct copies of p. The total number of normal classes over all vertical segments of $\hat{\alpha}$ is $2n$, since each vertical

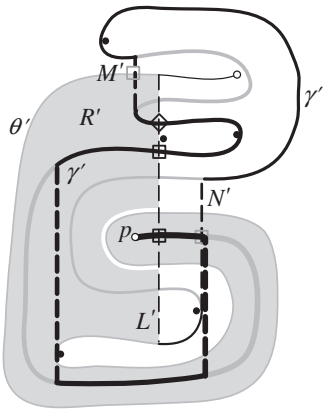

Fig. 12. $L \cap \gamma$ has normal intersections in two classes and special intersections.

segment contributes a unique contracted curve and at most two remaining fragments of α.

It remains to count special points that are self-intersections of $\hat{\alpha}$. In the remainder of the proof we show that each special self-intersection q can be mapped one-to-one to a normal class that contributes no normal self-intersection to $\hat{\alpha}$. (Thus the number of normal and special self-intersections is bounded by the number of normal classes.) The one special self-intersection in Fig. 12 (marked with a diamond) maps to the normal class of $M' \cap \theta'$.

Therefore, consider a special point $q \in L \cap \gamma$ that is a self-intersection of $\hat{\alpha}$. The lift γ' enters R' by crossing L' at q', but by the definition of special, γ' cannot end inside R'. Since γ' does not cross θ', it must leave by crossing L' again. If R' was to the right of L' at q', as in Fig. 11(a),(b), this would violate the left intersection property, so we know that R' lies to the left of L'.

The lift of $\hat{\alpha}$ must exit R' by a vertical segment M' to the left of L', as in Fig. 11(c),(d). This M' has its own region $R(M', \rho')$, for some $\rho \subset \gamma$. Since M' exits R' without crossing L', the curve θ' must cross M' an odd number of times, and therefore θ' ends inside $R(M', \rho')$.

Let ℓ denote the line through L' in the universal cover. Since θ' ends on ℓ inside $R(M', \rho')$, there is a simply connected subset of this region, defined by ℓ and ρ', that the lift α' enters from left to right at the endpoint of θ'. Examples are drawn with stripes in Fig. 11(c),(d). Since curve α' cannot cross ρ' and crossing ℓ again would violate the left intersection property, α' ends in this region, and the first intersection of M and θ is normal.

We can further observe that the intersections of $M' \cap \theta'$ form a normal class, because the other end of θ also crosses ℓ and cannot return to M' without violating the left intersection property. This normal class corresponds to at most one special self-intersection $q \in \hat{\alpha}$: the portion of α intersecting M contracts to the unique vertical segment L, and the monotone chain of $\hat{\alpha}$ that contains M contains at most one point $q \in L$. Since the normal class $M' \cap \theta'$ does not contribute a normal self-intersection, the total number of normal and special self-intersections is bounded by the maximum number of normal classes, which

is zero under the pin and pushpin definitions and $2n$ under the tack definition. This completes the lemma. □

We use an unrectified leftist path as our canonical path $\hat{\alpha}$ from now on.

3.6. *Comparing Canonical Paths*

Canonical sequences can have quadratic size, as the example of Fig. 2 shows, so we still cannot compute the sequences for leftist canonical paths $\hat{\alpha}$ and $\hat{\beta}$. Fortunately, these paths can be compared by a more complicated version of the sweep algorithm for rectifying paths.

We continue to use notation introduced in the previous section. Path $\hat{\alpha}$ is a sequence of monotone chains delimited by turn points: let p_0 be the path starting point, p_m the path endpoint, and p_i the ith turning point we encounter as we trace $\hat{\alpha}$. The ith monotone chain α_i is defined as the portion of the path $\hat{\alpha}$ from p_{i-1} to p_i. We first state two lemmas that will allow us to compare canonical paths, then we describe a sweep algorithm to perform the comparison.

Lemma 7. *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 endpoints 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 paths 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 a monotone chain, each monotone chain α_i and β_i has the same endpoints. □

Lemma 7 gives a necessary condition for $\hat{\alpha}$ and $\hat{\beta}$ to have the same canonical sequences: their monotone chains must have the same endpoints. 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.

We define *above sets* for points as follows: for each point p in the plane, let $A(\alpha, p) \subset \{1, 2, \ldots, m\}$ be the set of indices of monotone chains $\{\hat{\alpha}_i\}$ that lie above p. Similarly, let $A(\beta, p)$ be the set of indices of monotone chains $\{\hat{\beta}_i\}$ that lie above p.

Lemma 8. *If Lemma 7 is satisfied, then the paths $\hat{\alpha}$ and $\hat{\beta}$ have the same canonical sequence if and only if above sets $A(\alpha, p) = A(\beta, p)$ for each $p \in P$.*

Proof. To compare the canonical sequences, we can traverse the two paths $\hat{\alpha}$ and $\hat{\beta}$ with the same speed along the x-coordinate. Since the endpoints of each monotone chain $\hat{\alpha}_i$ are the same as $\hat{\beta}_i$, 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 $\hat{\alpha}_i$ and $\hat{\beta}_i$ 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 ($\overline{p}$ 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 $\hat{\beta}$. □

Lemma 8, with one more idea, allows us to compare the canonical sequences using a sweep algorithm. The input of the algorithm is the set of monotone α-chains from leftist path $\hat{\alpha}$ and monotone β-chains from $\hat{\beta}$. We sweep these chains from left to right with a vertical line. We maintain three invariants during the sweep, the most important being the *difference numbers* $D_x(k)$ defined below:

1. We know the set of monotone α-chains and the set of monotone β-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 $\hat{\alpha}$ and for $\hat{\beta}$ separate, otherwise we would need to compute all intersections between $\hat{\alpha}$ and $\hat{\beta}$.)
3. Finally, for sweep position x, let $A(\alpha, k)$ denote the *above set* of indices of monotone chains from $\hat{\alpha}$ that have ranks 1 through k. (That is, if we choose a point p on the sweepline just below the kth intersection with a monotone chain of $\hat{\alpha}$, then $A(\alpha, k) = A(\alpha, p)$. 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 8 to compare the canonical sequences. When the sweep encounters a point p, we locate p in the set of ordered monotone $\hat{\alpha}$-chains and the set of $\hat{\beta}$-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 8. 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 8 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 8.

Having shown that the invariants allow us to compare canonical sequences, we give algorithms to maintain these invariants and establish their 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 monotone α-chain indices, and T_β stores β-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, we briefly describe how to initialize these data structures for a given sweep position x.

Lemma 9. *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 $k = 1, \ldots, m$. First build two auxiliary tables: Let $I_\gamma(r)$ be the chain index at a given rank in T_γ and let $R_\gamma(i)$ 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 $k = 1, \ldots, m$, compute

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

This simply says that the difference number increases by one whenever the kth chain in the α list ranks higher in the β list and decreases when it ranks lower, and the same for the kth 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. □

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 endpoints 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 simplyfing assumptions about the monotone chains so that the algorithm is easier to describe—we remove these assumptions later. First, we treat each monotone chain as strictly monotone, even though chains from the canonical path may contain vertical line sements. Second, we 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 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 $D_x(k) = |\mathrm{diff}(A(\alpha, k), A(\beta, k))|$. 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 endpoint*. At a path endpoint, which must be shared by the α chain and β chain, we use Lemma 9 to reinitialize the data structures.

(IV) *Chain starting event*. At a turn point, we may have m monotone chains begin in α and in β. We first apply Lemma 9 just to the m new chains to compute the

difference numbers restricted only to the new chains, which we denote $D'_x(r)$, for $r = 1, \ldots, m$.

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. Note that the α and β chains will be spliced as whole bundles into their respective trees, because they start at the same turning point.

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 8. Thus, we need only insert m new differences; the old difference numbers do not otherwise change. Since the rth interval in the new monotone chains becomes the $(r + k)$th interval after the insertion, we insert the new difference numbers $\{D_x(r + k) = D'_x(r) + D_x(k), r = 1, 2, \ldots, m\}$ 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 $\{D_x(k + 1), D_x(k + 2), \ldots, D_x(k + m)\}$ from T_D.

The number of intersection events (type II) depends on the definition of path that we use, but we have shown a linear upper bound in the previous section. Thus, our sweep to compare canonical sequences is efficient.

Theorem 10. *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 bound 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 endpoint, and each operation for maintaining T_D (computing, insertion, or deletion of the difference numbers) can be charged either to a monotone chain endpoint or an intersection point. Since the time for each of these operations is at most $O(\log n)$, and we assume that there are a $O(n)$ number of monotone chain endpoints and intersection points, the total time is $O(n \log n)$. □

By careful implementation, we can eliminate 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 is from the right, it should be handled immediately before the vertical line segment.

4. Homotopy Test for Nonsimple 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 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 11. *A simple spanning tree of P with crossing number $O(\sqrt{n})$ can be constructed in $O(n^{1+\varepsilon})$ time, for any $\varepsilon > 0$.*

Proof. We combine ideas and results from [13], [18], [22], and [23]. Start by computing a spanning tree T_1 of P with crossing number $O(\sqrt{n})$. In [23] it is shown how a simplicial partition $\{P_1, \ldots, P_{n/2}\}$ with $|P_i| \leq 4$ and crossing number $O(\sqrt{n})$ can be constructed in time $O(n^{1+\varepsilon})$. This partition plays the role of the partial matching used in [22]: for each class P_i with $|P_i| \geq 2$, we put an edge between a pair of points lying in P_i, and remove either of its endpoints from the set of points P. Iterating this step $O(\log n)$ times, we get the spanning tree T_1 within $O(n^{1+\varepsilon})$ operations.

Tree T_1 can have self-intersections, but we can use it to construct a Steiner spanning tree T_2 that is simple [13]. 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 [18] 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 a subset of T_1, it also has crossing number $O(\sqrt{n})$.

As proved in [13], 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. □

Theorem 12. *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 [20]. Start with some vertex of α in B_i, where bit $i \in \{0, 1\}$, and create a tree node v 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 v and η, annotated with the edge e. If edge e has been crossed, then v has an edge annotated with e that connects to a node η. In either case, update the current node $v = \eta$, 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. □

5. Lower Bounds

In this section we establish lower bounds for the path homotopy question among points in the plane. The bounds are different depending on whether 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 $x_1, \ldots, x_n$, and a set of n disjoint intervals $[a_1, b_1], \ldots, [a_n, b_n]$, 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 $\Omega(n \log n)$.

However, 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 Fig. 13. If we consider the

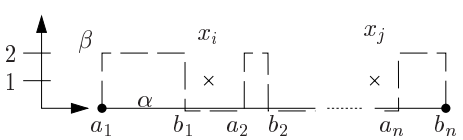

Fig. 13. Lower bound for simple paths.

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. *Nonsimple 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 [7]. 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 [16].

We give a reduction that works for any algorithm whose primitive tests check the signs of bounded-degree polynomials of the input point coordinates, which includes partition algorithms. 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 [14].

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, as illustrated in Fig. 14. 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 if and only if 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 nonsimple paths, we use standard machinery to give a subquadratic algorithm. Lower bounds show that the running time of our first algorithm is tight, but that it may be possible to improve the second.

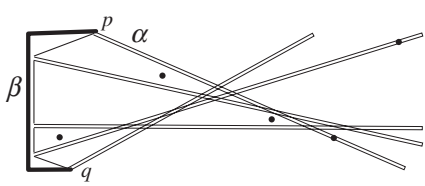

Fig. 14. Reduction from Hopcroft's problem.

Acknowledgements

The UNC authors thank Anna Lubiw for warning us about the intersections of the type in Fig. 10. The first author is grateful to Mark de Berg and Marc van Kreveld for helpful discussions regarding this research. He also thanks Pankaj Agarwal for pointing him to hierarchical cuttings to reduce the construction time in Lemma 11.

References

1. M. A. Armstrong. *Basic Topology*. McGraw-Hill, London, 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. S. Bespamyatnikh. Computing homotopic shortest paths in the plane. In *Proc. 14th ACM–SIAM Sympos. Discrete Algorithm*, pages 609–617, 2003.
5. B. Buttenfield. Treatment of the cartographic line. *Cartographica*, 22:1–26, 1985.
6. B. Chazelle. An algorithm for segment-dragging and its implementation. *Algorithmica*, 3:205–221, 1988.
7. M. de Berg and O. Schwarzkopf. Cuttings and applications. *Internat. J. Comput. Geom. Appl.*, 5:343–355, 1995.
8. 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.
9. T. K. Dey and S. Guha. Transforming curves on surfaces. *J. Comput. System Sci.*, 58:297–325, 1999.
10. 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.
11. J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. E. Tarjan. Making data structures persistent. *J. Comput. System Sci.*, 38:86–124, 1989.
12. H. Edelsbrunner. A note on dynamic range searching. *Bull. EATCS*, 15:34–40, 1981.
13. H. Edelsbrunner, L. J. Guibas, J. Hershberger, R. Seidel, M. Sharir, J. Snoeyink, and E. Welzl. Implicitly representing arrangements of lines or segments. *Discrete Comput. Geom.*, 4:433–466, 1989.
14. 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.
15. A. Efrat, S. G. Kobourov, and A. Lubiw. Computing homotopic shortest paths efficiently. In *Proc. ESA '2002*, pages 411–423, 2002.
16. J. Erickson. New lower bounds for Hopcroft's problem. *Discrete Comput. Geom.*, 16:389–418, 1996.
17. S. Gao, M. Jerrum, M. Kaufmann, K. Mehlhorn, W. Rülling, and C. Storb. On continuous homotopic one layer routing. In: *Proc. Fourth Annu. Sympos. Comput. Geom.*, pages 392–402, 1988.
18. 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.
19. J. Hershberger and J. Snoeyink. Computing minimum length paths of a given homotopy class. *Comput. Geom. Theory Appl.*, 4:63–98, 1994.
20. J. Hershberger and Subhash Suri. A pedestrian approach to ray shooting: shoot a ray, take a walk. *J. Algorithms*, 18:403–431, 1995.
21. F. M. Maley. *Single-Layer Wire Routing and Compaction*. MIT Press, Cambridge, MA, 1990.
22. 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.
23. J. Matoušek. Efficient partition trees. *Discrete Comput. Geom.*, 8:315–334, 1992.
24. J. R. Munkres. *Topology: A First Course*. Prentice-Hall, Englewood Cliffs, NJ, 1975.
25. L. Palazzi and J. Snoeyink. Counting and reporting red/blue segment intersections. *CVGIP: Graph. Models Image Process.*, 56(4):304–311, 1994.
26. F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*, 3rd edn. Springer-Verlag, New York, 1990.
27. H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.

Received July 16, 2002, *and in revised form April* 16, 2003. *Online publication November* 5, 2003.