Paired Pointset Traversal

Peter Hui
Department of Computer Science
DePaul University
243 South Wabash
Chicago, Illinois 60604, USA
phui@students.depaul.edu

Marcus Schaefer
Department of Computer Science
DePaul University
243 South Wabash
Chicago, Illinois 60604, USA
mschaefer@cs.depaul.edu

February 27, 2004

Abstract

In the Paired Pointset Traversal problem we ask whether, given two sets $A = \{a_1, \dots, a_n\}$ and $B = \{b_1, \dots, b_n\}$ in the plane, there is an ordering π of the points such that both $a_{\pi(1)}, \dots, a_{\pi(n)}$ and $b_{\pi(1)}, \dots, b_{\pi(n)}$ are self-avoiding polygonal arcs? We show that Paired Pointset Traversal is **NP**-complete. This has consequences for the complexity of computing the Fréchet distance of two-dimensional surfaces.

1 Introduction

Suppose we are given two sets of points $A = \{a_1, \dots, a_n\}$ and $B = \{b_1, \dots, b_n\}$ in the plane. How hard is it to determine whether we can traverse both A and B in the same order such that the resulting polygonal arcs do not self-intersect? Phrased differently, is there a permutation π such that both $a_{\pi(1)}, \dots, a_{\pi(n)}$ and $b_{\pi(1)}, \dots, b_{\pi(n)}$ are self-avoiding polygonal arcs? We call this problem the Paired Pointset Traversal problem. We will show that deciding Paired Pointset Traversal is **NP**-complete.

This problem is of interest in relation to a notion from geometry, the Fréchet distance. Intuitively, the Fréchet distance between two geometric objects is the largest distance between two points in a simultaneous "smooth" traversal of both objects. For example, for one-dimensional curves $f : [0, 1] \to \mathbb{R}$ and $g : [0, 1] \to \mathbb{R}$, the Fréchet distance is

$$\delta_F(f, g) = \inf_{\alpha,\beta} \max_{s\in[0,1]} d(f(\alpha(s)), g(\beta(s))),$$

where α and β range over all continuous parameterizations of $[0, 1]$. The Fréchet distance captures, better than the more familiar Hausdorff distance does, the notion of similarity in shape.

The Fréchet distance between two polygonal arcs can be computed in polynomial time [1]. On the other hand, the **NP**-completeness of Paired Pointset

Traversal implies that approximating the Fréchet distance of two-dimensional surfaces by sampling is **NP**-complete, since, after picking the sample points at random, we still have to determine whether there is a simultaneous traversal, which is exactly the Paired Pointset Traversal problem.[1]

We also consider a variant of the Paired Pointset Traversal problem, in which instead of straight lines we allow curves to connect the points, but restrict the curves to lie within the convex hull of their pointsets. This version can be solved in linear time.

2 Paired Pointset Traversal

We formally define our problem Paired Pointset Traversal as follows:

INSTANCE: Two planar pointsets $A = \{a_1, a_2, \ldots, a_n\}, B = \{b_1, b_2, \ldots, b_n\}$.

QUESTION: Is there a permutation π such that $(a_{\pi(1)}, a_{\pi(2)}, \ldots, a_{\pi(n)})$ and $(b_{\pi(1)}, b_{\pi(2)}, \ldots, b_{\pi(n)})$ both form self-avoiding polygonal paths?

Note that we do not prevent the two paths $(a_{\pi(1)}, a_{\pi(2)}, \ldots, a_{\pi(n)})$ and $(b_{\pi(1)}, b_{\pi(2)}, \ldots, b_{\pi(n)})$ from intersecting *each other*— the only restriction that we place upon the paths is that neither path may intersect itself. We refer to such paths as *matched self-avoiding paths.*

Theorem 2.1 Paired Pointset Traversal *is* **NP**-*complete.*

Proof We show **NP**-hardness by a reduction from the following version of Hamiltonian Path:

INSTANCE: A plane graph $G = (V, E)$ (that is a graph embedded in the plane without intersections), and two of its vertices, s and t.

QUESTION: Does G contain a Hamiltonian path from s to t.

This problem is well-known to be **NP**-complete [2], even if we assume that s and t are on the boundary of the convex hull formed by V (an assumption we will need later).

Given the the plane graph $G = (V, E)$ and two vertices s and t, we construct vertex sets $A = \{a_1, a_2, \ldots, a_n\}$ and $B = \{b_1, b_2, \ldots, b_n\}$ in the plane such that there is Hamiltonian path from s to t in G if and only if A and B allow a matched self-avoiding path. This immediately shows that Paired Pointset Traversal is **NP**-complete.

Let $G = (V, E)$ be a plane graph with two special vertices s and t. We start by adding the vertices of V to both A and B. We refer to these vertex sets as V_A and V_B, respectively. Then, for each vertex pair $(u, v) \notin E$, we select a point on the straight line $\overline{uv}$ and add it to B, making sure that the point does not lie

[1]This connection was pointed out to us by Binhai Zhu, who also supplied us with the problem in the first place [4].

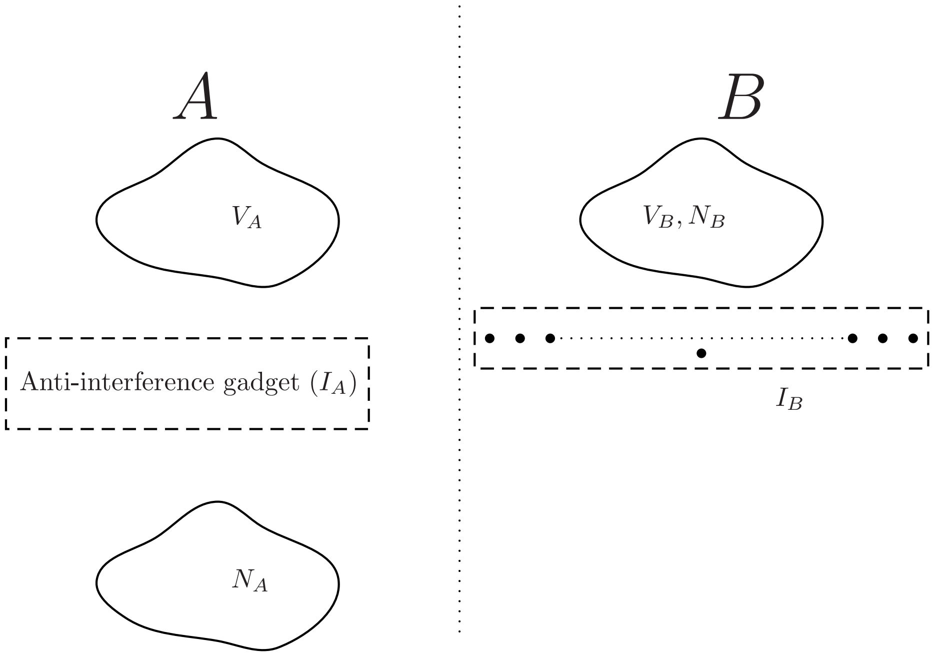

Figure 1: Reduction from Planar Hamiltonian Path.

on any other straight line $\overline{ab}$ with $a, b \in V$; we call the set of these vertices N_B. These points will assure that we cannot traverse B by going directly from u to v for any edge (u, v) that is absent from E. We then add a corresponding set of vertices, N_A to A. The vertices in N_A have the same relative position to each other as the vertices in N_B, but they appear translated significantly beneath the vertices of V_A so as to allow space for another gadget that will be placed between V_A and N_A. We also add to N_A and N_B another set T_A and T_B, which we will specify later in the proof. T_A and T_B will be necessary to ensure that a simultaneous travererersal is possible.

Finally, we need a third set of vertices to prevent traversals of A that include edges going back and forth between V_A to N_A. More precisely, we will add sets I_A and I_B such that any pair of matched self-avoiding paths of A and B will (i) have to start in I_A (I_B) and traverse all of I_A (I_B) with the exception of a single vertex i_A (i_B), then (ii) traverse all of V_A (V_B), (iii) go to i_A (i_B), and finally (iv) traverse all of N_A (of course the traversal could be in the reverse order). We will show how to construct this anti-interference gadget later, and first prove the correctness of the construction. Figure 1 illustrates the construction up to this point.

Assuming then that the interference gadget works correctly, and we can choose T_A and T_B to make a traversal of N_A and N_B possible, we can prove the correctness of the construction. First suppose that there is a Hamiltonian path in G. We can then traverse A and B simultaneously as follows. Begin with $I_A - \{i_A\}$ ($I_A - \{i_A\}$). Continue to s, and follow a Hamiltonian path through

V_A (V_B) to t. From t go to i_A (i_B), and then on to N_A (N_B). We will show later how to build N_A and N_B such that they can be traversed simultaneously. For the other direction, suppose there is a pair of matched self-avoiding path. Since we assumed that the anti-interference gadget works in that case, we know that the paths traverse all of V_A (V_B) before they begin traversing N_A (N_B). Furthermore, The vertices in N_B ensure that the path through V_B does not use any edges outside of E. Hence, the traversal of V_B constitutes a Hamiltonian path in G, which is what we had to show.

We still owe the reader two details of the construction: how to build the anti-interference gadget, and how to ensure that N_A and N_B can be traversed simultaneously.

Let us first show how to build the anti-interference gadget I_A and I_B. Our goal is to arrange the vertices of I_A and I_B such that all the points in V_A (V_B) get traversed before the points in N_A (N_B), or vice versa. We accomplish this by arranging the vertices of I_A in a series of nested wedges, and the vertices of I_B in a line (see Figure 2).

We claim that there is only one simultaneous traversal of I_A and I_B, namely the one shown in the figure: I_A must be traversed starting at a_1, continuing outwards forming a series of nested wedges to a_{k+1}. If the claim is true, this means that the anti-interference gadget performs its function: it separates the traversal of V_A from the traversal of N_A, and, furthermore, its vertices do not interfere with the traversals of V_A and N_A.

We will establish the claim in several steps. We choose $k = 5(|V_B|+|N_B|)+4$. Consider the vertices along the line $b_1, b_2, \ldots, b_{k-1}$. These vertices can only be traversed in order, unless a traversal uses a vertex outside of that line. Now there are at most $|V_B| + |N_B| + 3$ vertices other than $b_1, b_2, \ldots, b_{k-1}$. This means a traversal must contain four neighboring vertices $(b_i, b_{i+1}, b_{i+2}, b_{i+3})$ (since k is large enough). This, in turn implies that I_A contains a wedge, namely (a_i, a_{i+1}, a_{i+2}) if i is odd, or $(a_{i+1}, a_{i+2}, a_{i+3})$ if i is even. The subsequent vertices, $a_{i+3}, \ldots, a_{k-2}$ (or $a_{i+4}, \ldots, a_{k-2}$ as the case might be), are entirely shielded from N_A and V_A, which means, since the corresponding b vertices form a line, that they have to be traversed in order, ending with a_{k-2}. The arrangement of the vertices now ensures that a_{k-1} is the only point that we can connect to next. Therefore, (b_{k-2}, b_{k-1}) must be present as well.

Now, (b_{k-2}, b_{k-1}) separates b_k from V_B and N_B.

We must now show that from a_{k-1}, the only possible next vertex is a_k.

1. We cannot proceed from a_{k-1} to a_{k+1}, since doing so would result in a self-intersection in B.

2. We cannot proceed from a_{k-1} to any vertex in V_A, since doing so would eventually require one of the following next steps:

   (a) *Moving to a_{k+1}.* From a_{k+1} we could then only continue on to V_A, since N_A, and i_A are shielded from it. From V_A we would then have to continue to i_A. At this point, both b_k and the vertices of N_B have yet to be visited. We have already proven that we cannot traverse

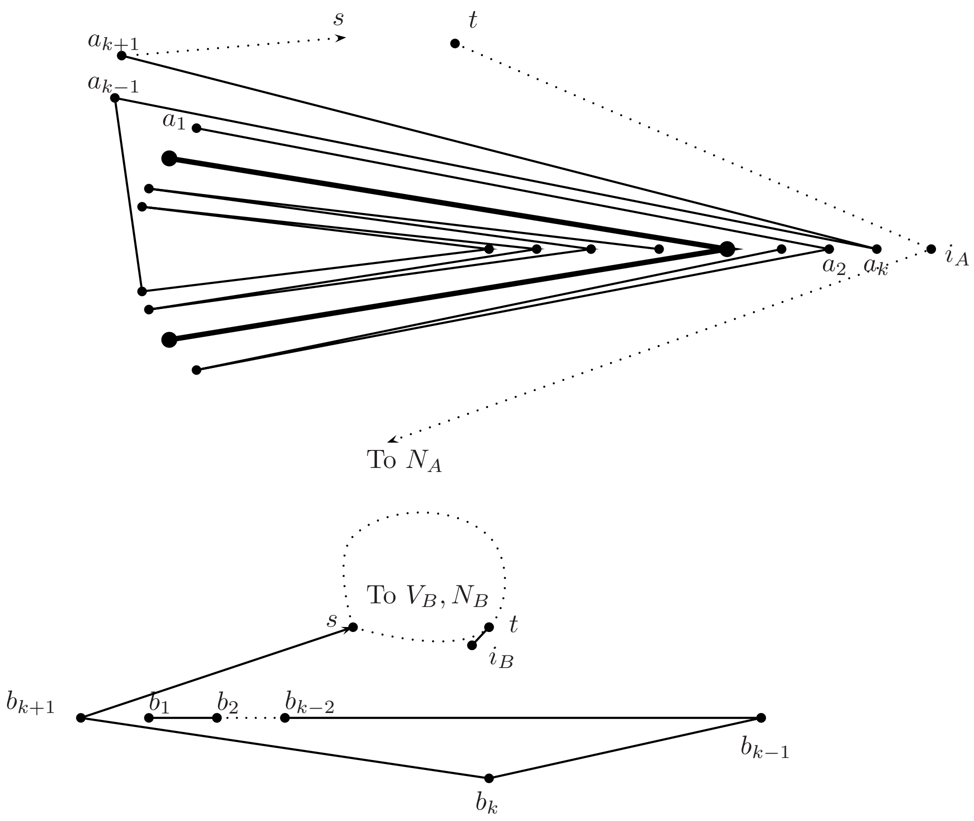

Figure 2: Anti-interference gadget for pointsets A (top) and B (bottom), along with their sole legal traversal.

between b_k and N_B, so regardless of whether we decide to visit b_k or N_B, it will be impossible to visit the other.

(b) *Moving to* i_A. At this point we can continue to either N_A or $V_A \cup \{a_{k+1}\}$, but we can no longer traverse both.

3. We cannot proceed from a_{k-1} to i_A, since doing so would make it impossible to traverse both V_A and N_A (as above).

This leaves a_k as the sole possible next vertex from a_{k-1}, which in turn implies that the vertices of I_A are traversed as a complete sequence of nested wedges (since (a_{k-1}, a_k) separates V_A from the wedges.

Finally, we have to show how to wire N_A and N_B such that a simultaneous traversal is possible. Remember that N_B contains vertices chosen to obstruct edges that do not occur in E. Now once V_A and V_B (and the anti-interference device) have been traversed, we need to pick out these vertices. To this end, we

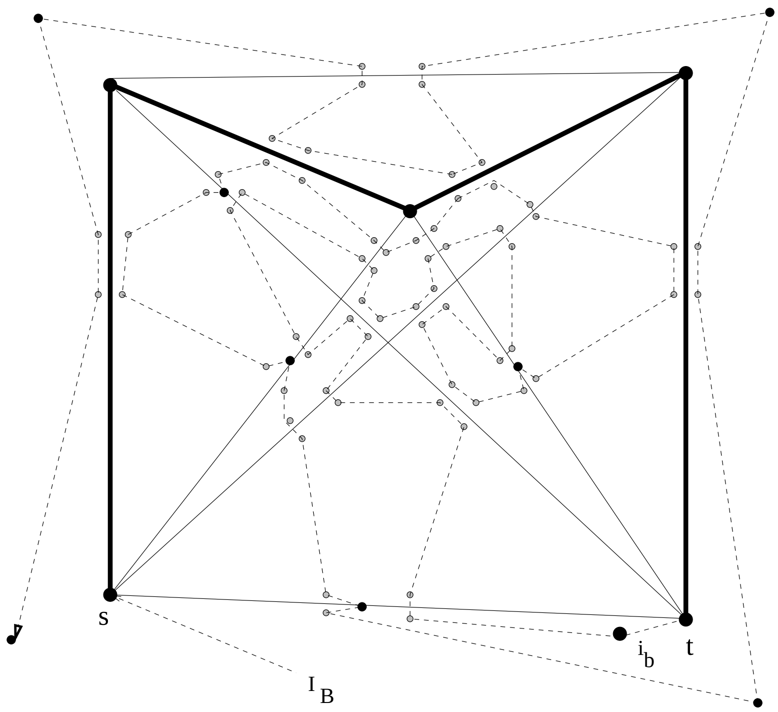

Figure 3: A graph containing a Hamiltonian path (in bold) with vertices N_B added, along with its traversal (dotted).

think of the area of V_B being separated into polygonal regions by the complete graph (drawn using straight lines) on the vertices V_B. To each resulting line segment we add four vertices, two on each side of the line segment. Furthermore, if the line segment contains an obstruction point, we make sure it is in the middle between the new vertices. Finally, we add vertices surrounding V_A, so we can pick up any region that is not visible from I_A. Figure 3 illustrates the construction. It shows a particular Hamiltonian path having been chosen (in bold), and, based on that, the subsequent traversal of N_B to pick up all remaining vertices. Since N_A is a translation of N_B, and does not interact with V_A, we can traverse the vertices in the order shown. The main observation is that a path cannot make any regions unreachable, since it is not closed.

This completes the construction. □

3 The Convex Hull Case

Next, we consider a version of the problem in which we allow arbitrary curves (rather than straight lines) between points. This change by itself would make the pointset problem trivial: there always is a solution. However, the situation changes, if we require the curves to be contained within the convex hull formed by their respective pointsets. In that case, a simultaneous traversal is not always possible.

While the initial problem is **NP**-complete, we show that the convex hull variant can be solved in linear time using a combination of greedy choice and dynamic programming.

We first present an algorithm which solves an important special case of the problem, in which all points lie on the boundary of the convex hull of their respective pointsets. We will show how to solve this special case in quadratic time using greedy choice, and then improve it to linear time with the help of dynamic programming. Finally, we will show how to deal with interior vertices.

3.1 The Hollow Hull Case

3.1.1 Preliminaries

We first consider the case where both pointsets contain only points which lie along the boundary of the convex hull of their respective pointsets (see Figure 4).

We begin with a few definitions and observations:

Definition 3.1 *We define the* clockwise ordering $(v_1, v_2, \ldots, v_n)$ starting at vertex v_1 *of the vertices in a convex pointset as the order in which the points would be visited if the pointset were traversed along the boundary of the convex hull in clockwise order, starting at vertex* v_1.

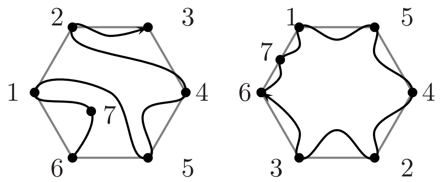

Figure 4: Two pointsets, A and B, with a legal traversal (6 7 1 5 4 2 3) of the two pointsets. We allow arbitrary curves between points, provided that all curves lie within the convex hulls of their respective pointsets.

Next, we introduce a notation to express that a point in A corresponds to a point in B:

Definition 3.2 *We say* a_i corresponds to b_i *and write* $a_i \equiv b_i$ *for* $a_i \in A, b_i \in B$ *and all* i.

Our first observation is that if we start with the clockwise ordering of a convex pointset, and continue to remove either the first or last vertex in that ordering, the resulting sequence of selected vertices will yield a self-avoiding path.

Lemma 3.3 *Given a convex pointset, and a corresponding clockwise ordering of that pointset, the path induced by repeatedly selecting and removing either the first or last vertices, and repeating the same process for as long as possible on the resulting clockwise ordering of the remaining vertices will be self-avoiding.*

We skip the proof for reasons of space.

Our second observation is that if a self-avoiding path starting at v exists, then it is possible to "reconstruct" that path by repeatedly selecting and removing either the first or last vertices in the remaining clockwise ordering, beginning with the clockwise ordering starting at v. First, we define the notion of a path's reverse:

Definition 3.4 *Let* $P = (v_1, v_2, \ldots, v_m)$ *be a path or portion thereof through some planar pointset. Then we define* $P^{\mathcal{R}} = (v_m, v_{m-1}, \ldots, v_2, v_1)$.

Lemma 3.5 *Given a self-avoiding path* $P = (v_{\pi(1)}, v_{\pi(2)}, \ldots, v_{\pi(n)})$ *through a convex pointset, we can derive* P *by applying the algorithm outlined in Lemma 3.3 beginning with the clockwise ordering of the pointset starting at* $v_{\pi(1)}$.

Proof Assume for a contradiction that at some iteration i of the algorithm outlined in Lemma 3.3, in order to construct the self-avoiding path P, we are required to select a vertex other than either the first or last in the remaining clockwise ordering. Let v_x be the vertex selected at iteration $i-1$, and v_y the vertex that must be selected at iteration i in order to construct the self-avoiding path. Let $(v_{y-m}, \ldots, v_{y-1}, v_y, v_{y+1}, \ldots, v_{y+n})$ be the remaining clockwise ordering from which we must select the remaining vertices to complete P. Observe that by including (v_x, v_y) in P, we have partitioned the remaining vertices into two disjoint subsets $V_l = \{v_{y-m}, \ldots, v_{y-1}\}$ and $V_r = \{v_{y+1}, \ldots, v_{y+n}\}$. Furthermore, since P is a path, there must exist at least one vertex pair (v_l, v_r) such that $v_l \in V_l, v_r \in V_r$. But since v_l and v_r lie on opposite sides of $\widetilde{v_x v_y}$, the curves $\widetilde{v_x v_y}$ and $\widetilde{v_l v_r}$ intersect, and we obtain a contradiction. $\square$

Corollary 3.6 *If a path* P *is derivable from the clockwise ordering of a planar pointset* A *using the algorithm outlined in Lemma 3.3, then so is its reverse* $P^{\mathcal{R}}$.

Our final observation is that at any point in the algorithm, if we have a choice between two vertices to select, then choosing one of the vertices will enable us to complete a matched self-avoiding path if and only if choosing the other will as well.

Lemma 3.7 *Let*

$$\begin{aligned} A &= (a_1, a_2, \ldots, a_n), \text{ and} \\ B &= (b_{\pi(1)}, b_{\pi(2)}, \ldots, b_{\pi(n)}) \end{aligned}$$

be two clockwise orderings encountered during the execution of the algorithm outlined in Lemma 3.3, where $a_1 \equiv b_{\pi(1)}$ *or* $a_1 \equiv b_{\pi(n)}$, *and* $a_n \equiv b_{\pi(1)}$ *or* $a_n \equiv b_{\pi(n)}$. *Choosing* $a_1 (\equiv b_{\pi(1)})$ *yields a self-avoiding path if and only if choosing* $a_n (\equiv b_{\pi(n)})$ *does as well.*

Proof We prove by contradiction. There are two cases to consider: the case where $a_1 \equiv b_{\pi(1)}$, and the case where $a_1 \equiv b_{\pi(n)}$. In the first case, we note that by choosing and eliminating $a_1 \equiv b_{\pi(1)}$, we are left with the remaining clockwise orderings $(a_2, \ldots, a_n)$ and $(b_{\pi(2)}, \ldots, b_{\pi(n)})$. If it is the case that $a_2 \equiv b_{\pi(2)}$, then we can safely and legally match and remove these two as well, without fear of jeopardizing the other end of the ordering. We can repeat this process of matching and removing vertices from the left ends of the remaining orderings until we are left with a pair of unmatched vertices $a_i \not\equiv b_{\pi(i)}$. We can then repeat from the right side. Note that regardless of the order in which we truncate from the edges, we will eventually reach the same configuration, so the order in which the vertices were selected and removed is inconsequential with respect to the ability to created a pair of matched self-avoiding paths. We thus have a contradiction.

Formally, let $A = (a_1, \ldots, a_n)$ and $B = (b_{\pi(1)}, \ldots, b_{\pi(n)})$ be the clockwise orderings obtained by repeatedly selecting and removing each successive pair of leftmost vertices a_l and $b_{\pi(l)}$ so long as $a_l \equiv b_{\pi(l)}$. At this point, the leftmost vertices $a_l \not\equiv b_{\pi(l)}$. From this configuration, the only remaining choice is to select and remove a_n and $b_{\pi(n)}$, and from here, we can repeatedly select and remove the rightmost vertices a_r and b_r so long as $a_r \equiv b_{\pi(r)}$, until we are left with a configuration such that the right ends no longer match. We are thus left with $A = (a_l, \ldots, a_r)$ and $B = (b_{\pi(l)}, \ldots, b_{\pi(r)})$, from which we know by our initial assumption that we can continue on to complete a self-avoiding path.

Now, assume that our initial selection was vertex $a_n(\equiv b_{\pi(n)})$ instead. We know that after selecting a_n, we can continue removing from the left and right ends of A and B as above, and that by doing so, we will ultimately end up in the same configuration $A = (a_l, \ldots, a_r)$ and $B = (b_{\pi(l)}, \ldots, b_{\pi(r))}$. However, by our assumption, we have that by continuing from this point, we ultimately cannot derive a self-avoiding path, and we have derived a contradiction.

The case where a_1 initially matches with $b_{\pi(n)}$ instead of $b_{\pi(1)}$ follows along the same lines, as does the case in which we choose a_n before a_1. □

At this point, we can write an algorithm which, given index i and length l, determines in linear time whether or not a paired self-avoiding path fragment of length l exists traversing A and B and starting at a_i and b_i (see Algorithm 4 in the appendix). We could then extend this algorithm by a brute force approach and try each $(a_i, b_i), 1 \leq i \leq n$, to determine whether any such *complete* path exists in quadratic time. However, with a slight extension of this algorithm, we can tighten this bound even further to make the entire algorithm run in linear time. We do this by using dynamic programming. We essentially attempt to build segments of a path starting at an arbitrary $(a_{\pi(i)}, b_{\pi(i)})$, saving the computed path for subsequent attempts at a different $(a_{\pi(j)}, b_{\pi(j)})$ in the event that this one fails. In this manner, no subpath will ever need to be recomputed. Intuitively, we can start our attempt to build a path using Algorithmm 4, proceeding until there are no available vertices to add to our path. At this point, we are still guaranteed that the path constructed so far is self-avoiding. However, instead of giving up, we can store the already computed path segment for later reference while trying a different start vertex. Our next (somewhat trivial) observation is that after trying a different start vertex, if we manage to select either of the endpoints of any previously computed segment, we can merge the two segments and obtain a larger self-avoiding path. As a formality, we first define the notion of the concatenation of two paths:

Definition 3.8 *Let $P_1 = (v_i, v_{i+1}, \ldots, v_n)$ and $P_2 = (v_j, v_{j+1}, \ldots, v_m)$ be two paths through some planar pointset. We define $P_1 \circ P_2$ to be the path*

$$(v_i, v_{i+1}, \ldots, v_n, v_j, v_{j+1}, \ldots, v_m)$$

through the same pointset.

Lemma 3.9 *Let $A_1 = (u_1, \ldots, u_k)$ and $B_1 = (v_1, \ldots, v_k)$ be a pair of self-avoiding paths obtained by repeatedly pairing, selecting and removing either the first or last vertex from each successive remaining clockwise ordering, starting from the clockwise orderings of A and B starting at u_1 and v_1, respectively. Let $A_2 = (u_k, \ldots u_n)$ and $B_2 = (v_k, \ldots, v_n)$ be another set of such paths through the same pointsets obtained in the same manner. Both $A_1 \circ A_2$ and $B_1 \circ B_2$ are self-avoiding paths.*

Proof Since A_2 is a self-avoiding path, then it follows that the path $(u_k, \ldots, u_n)$ is inducible by applying Algorithm 1 to the clockwise ordering of the vertices of A starting at a_j. Similarly for B_2. Since u_k is the only vertex shared by A_1 and A_2, it must be the case that none of $(u_{k+1}, \ldots, u_n)$ were selected from any of the intermediate clockwise orderings during the execution of Algorithm 1 in deriving A_1, and hence at the algorithm's termination, these same vertices are still in the remaining clockwise ordering. We already know that we can continue on to construct a self-avoiding path $(u_k, \ldots, u_n)$ from the remaining clockwise ordering, so it immediately follows that $A_1 \circ A_2$ is a self-avoiding path. The same argument follows for $B_1 \circ B_2$. □

For the reverse direction, we show that if there exists some self-avoiding path through the vertices of A and B, then it can be reconstructed by repeatedly invoking Algorithm 4 on the vertices and merging the resulting fragments. There are three key points to observe here. First, if such a pair of paths exists, then all vertices in both pointsets must be included. Second, the clockwise orderings of A and B starting at a_i and b_i are fixed, so the construction of each path fragment has no detrimental effect on the ability to construct subsequent fragments. Third, once a legal path fragment is constructed, it remains legal regardless of any subsequent fragments which are appended or prepended to it. Note that this is only true because the overall clockwise orderings of A and B are fixed.

The first point is important in the sense that if, by blindly choosing a pair of vertices (a_i, b_i) at which to start Algorithm 4, we fail (i.e., we hit a dead end), then as long as some legal path exists, then the fact that we failed is only indicative of the fact that no path exists *starting at* (a_i, b_i), and that there is still hope as long as we choose a different start vertex.

The second point is important since it ensures that the *order* in which we attempt our start vertices is irrelevant. That is, initially choosing (a_i, b_i) as our start vertices for example, and retaining and reusing the subsequent path fragment, will not inadvertently invalidate subsequent attempts at constructing a pair of paths at a different pair of vertices (a_j, b_j). The reason why we can safely make this point is that for a given path fragment, subsequent choices are severely restricted; that is, the only possible next choices are the two endpoints of its remaining clockwise ordering, and as such, choosing either of these endpoints as its next vertex can only serve to expand its path, and not to alter the fate of any other attempted fragments investigated later.

The third point is important because it ensures that by merging two self-avoiding fragments, the resulting larger fragment will still be self-avoiding. Obviously, the two individual fragments will not suddenly self-intersect, so it only remains to preclude the possibility of interference between the two fragments. By Lemma 3.5, if the resulting fragment is self-intersecting, then it follows that the full fragment itself is not obtainable by applying Algorithm 4 to A starting at the appropriate start vertex. However, it is easy to show that this full fragment *is* obtainable in this manner, simply by following the selection order used to obtain the first fragment followed by the selection order used to obtain the second fragment. Combined with Corollary 3.6, this yields our claim. □

Together with Lemma 3.9, we arrive at a linear time algorithm for determining whether any paired self-avoiding path exists through the vertices of A and B (see Algorithm 4 in the appendix).

3.2 Non-hollow Hull Case

We now consider the case where the interior of the convex hull of at least one of the pointsets contains at least one point. We define $I(X)$ to be the set of points of X which do not lie on the boundary of the convex hull of X; for each point a_x that is in $I(A)$, and for each point b_y that is in $I(B)$, we add all of a_x, b_x, a_y, and b_y to a set $I(A, B)$:

$$
\begin{aligned}
I_A(A,B) &= \{a_i : a_i \in I(A) \text{ or } b_i \in I(B)\} \\
I_B(A,B) &= \{b_i : a_i \in I(A) \text{ or } b_i \in I(B)\}
\end{aligned}
$$

We show that instances where one or both pointsets contain at least one interior point reduce to the hollow hull case. That is, that a self-avoiding path through A and B exists if and only if one exists through $A - I_A(A, B)$ and $B - I_B(A, B)$ as well.

Before we continue, we note that since we connect points using arbitrary curves contained within the pointset's convex hull, we can place interior points *anywhere* within the interior of the pointset's convex hull without fear of invalidating existing paths or creating new ones where previously none existed.

There are two cases to consider. In the case where a_i and b_i are both interior points in their respective pointsets, it is trivial to observe that a matched self-avoiding path exists through $A - \{a_i\}$ and $B - \{b_i\}$ if and only if one exists through A and B as well.

In the case where a_i is interior to A, while b_i lies on the boundary of B's convex hull, we observe that if a matched self-avoiding path exists through $A - a_i$ and $B - b_i$, then we can add b_i back into B in its original position along the boundary of B's convex hull. Once the vertex has been replaced, we can proceed to divert the path through b_i legally, and place a_i in the interior of A in the corresponding position in the path. The case where b_i is internal and a_i is not follows similarly.

This shows that we can solve the general case, allowing interior points, in linear time, assuming we know the interior points. If this information is not given as part of the problem, we first have to spend time $O(n \log n)$ to determine the convex hulls of pointsets A and B.

4 Conclusion

In the hollow hull case, the following questions suggest themselves: how fast can we solve the paired pointset traversal problem if we allow up to k intersections. Is the resulting problem **NP**-complete? If so (and we conjecture that this is the case), how does the complexity vary with k? More precisely, taking the parameterized complexity view, can we show that the paired pointset traversal problem can be solved in time $O(n^c)$ for any fixed k, where c does not depend on k. A recent, comparable result, of Martin Grohe shows that we can determine in quadratic time, whether the crossing number of a graph is at most k (for any fixed k) [3].

Acknowledgment. We would like to thank Binhai Zhu for suggesting the problem to us, and explaining its application to measuring similarity of two-dimensional surfaces.

References

[1] Helmut Alt and Michael Godau. Measuring the resemblance of polygonal curves. In *Proceedings of the 8th Annual Symposium on Computational Geometry*, pages 102–109. ACM Press, 1992.

[2] M. R. Garey, D. S. Johnson, and R. Endre Tarjan. The planar Hamiltonian circuit problem is NP-complete. *SIAM Journal on Computing*, 5(4):704–714, December 1976.

[3] Martin Grohe. Computing crossing numbers in quadratic time. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, pages 231–236, July 6–8 2001.

[4] Binhai Zhu, 2004. Personal Communication.

Algorithm 1 Linear-time algorithm for deciding if a matched self-avoiding path of length l through A and B exists starting at points a_i and b_i

```
PATH ← λ
C_A ← ClockwiseOrdering(A, a_i)
C_B ← ClockwiseOrdering(B, b_i)
while (|PATH| < l) do
  if ((C_A[first] = C_B[first]) OR (C_A[first] = C_B[last])) then
    newVertex ← C_A[first]
    add newVertex to PATH
    delete newVertex from C_A
    delete newVertex from C_B
  else if ((C_A[last] = C_B[first]) OR (C_A[last] = C_B[last])) then
    newVertex ← C_A[last]
    add newVertex to PATH
    delete newVertex from C_A
    delete newVertex from C_B
  else
    return false {cannot proceed any farther }
  end if
end while
return true {both orderings have been consumed}
```

Algorithm 2 Linear time algorithm for deciding if a paired self-avoiding path exists between all vertices of A and B.

$PATH \leftarrow \lambda$
for $i = 1$ to $|A|(= |B|)$ **do**
  $PartialPaths[i] \leftarrow \lambda$
end for
$C_A \leftarrow$ ClockwiseOrdering(A, a_i)
$C_B \leftarrow$ ClockwiseOrdering(B, b_i)
loop
  if $|C_A| = |C_B| = 0$ **then**
    return true {*both orderings have been consumed*}
  end if
  if ((C_A[first] $= C_B$[first])**OR** (C_A[first] $= C_B$[last])) **then**
    if $PartialPaths[C_A$[first]$] \neq \lambda$ **then**
      $PATH = PATH \circ PartialPaths[C_A$[first]$]$ {Reuse previously computed paths}
    else
      $newVertex \leftarrow C_A$[first] {No previously computed data to reuse}
      add $newVertex$ to $PATH$
    end if
    $PartialPaths[PATH$[first]$] \leftarrow PATH$ {Update previously computed paths}
    $PartialPaths[PATH$[last]$] \leftarrow PATH^{\mathcal{R}}$
    for each (i in $PATH$) **do**
      remove i from C_A
      remove i from C_B
    end for
  else if ((C_A[last] $= C_B$[first])**OR** (C_A[last] $= C_B$[last])) **then**
    if $PartialPaths[C_A$[last]$] \neq \lambda$ **then**
      $PATH = PATH \circ PartialPaths[C_A$[last]$]$ {Reuse previously computed paths}
    else
      $newVertex \leftarrow C_A$[last] {No previously computed data to reuse}
      add $newVertex$ to $PATH$
    end if
    $PartialPaths[PATH$[first]$] \leftarrow PATH$ {Update previously computed paths}
    $PartialPaths[PATH$[last]$] \leftarrow PATH^{\mathcal{R}}$
    for each (i in $PATH$) **do**
      remove i from C_A
      remove i from C_B
    end for
  else
    if all vertices exhausted **then**
      return false
    else
      $C_A \leftarrow$ ClockwiseOrdering$(\{a \in C_A\}, C_A$[second]$)$
      $C_B \leftarrow$ ClockwiseOrdering$(\{b \in C_B\}, C_A$[second]$)$
    end if
  end if
end loop