

# Matching Planar Maps

Helmut Alt\*

Alon Efrat†

Günter Rote\*

Carola Wenk†

## Abstract

The subject of this paper are algorithms for measuring the similarity of patterns of line segments in the plane, a standard problem in, e.g. computer vision, geographic information systems, etc. More precisely, we will define feasible distance measures that reflect how close a given pattern  $H$  is to some part of a larger pattern  $G$ . These distance measures are generalizations of the well known Fréchet distance for curves. We will first give an efficient algorithm for the case that  $H$  is a polygonal curve and  $G$  is a geometric graph. Then, slightly relaxing the definition of distance measure we will give an algorithm for the general case where both,  $H$  and  $G$ , are geometric graphs.

## 1 Introduction

Patterns consisting of line segments occur in many applications of a geometric nature, like computer vision, geographic information systems, CAGD, etc. In many cases the problem occurs to determine whether some given pattern  $H$  is equal to or similar to some part of a larger pattern  $G$ . Here, for the case of patterns consisting of straight line segments, we will give feasible distance measures reflecting this similarity and being compatible to paths on the pattern. Also, we will give efficient algorithms for computing these distances.

As a first task we consider a given polygonal curve, and an embedded graph with line segment edges, and we wish to find a path in the graph (which then corresponds to a polygonal curve) such that the Fréchet distance between the curve and the path is minimized. This is a partial matching variant. The problem in this form already has many applications. The following one, for example, looked particularly appealing to us: The Global Positioning System (GPS) is a collection of satellites that provides worldwide positioning information. A specific position can be determined by using a GPS receiver. Now consider a given roadmap, and a person travelling on some of the roads, while recording its information using a GPS receiver. The roadmap

can be modelled by a planar embedded graph, and the path the person travelled is represented by a sequence of *GPS positions* recorded by the GPS receiver, which we connect by straight line segments to form a polygonal curve. Since the GPS receiver usually introduces noise, the captured curve will not exactly lie on the roadmap. The task is to identify those roads which have actually been travelled. This is a prerequisite for incrementally constructing roadmaps from such GPS curves, which is especially interesting for roads such as hiking trails in a forest which are not visible on aerial pictures. We present an algorithm solving this problem in Section 2. It has been implemented, and even without specific optimizations it runs surprisingly fast. In Section 3 we consider the case of two geometric graphs.

Our distance measures are based on the Fréchet distance for curves which has been investigated before in [1].

**DEFINITION 1.1. (FRÉCHET DISTANCE)** *Let  $f : I = [l_I, r_I] \rightarrow \mathbb{R}^2$ ,  $g : J = [l_J, r_J] \rightarrow \mathbb{R}^2$  be two planar curves, and let  $\|\cdot\|$  denote the Euclidean norm. Then the Fréchet distance  $\delta_F(f, g)$  is defined as*

$$\delta_F(f, g) := \inf_{\substack{\alpha: [0,1] \rightarrow I \\ \beta: [0,1] \rightarrow J}} \max_{t \in [0,1]} \|(f(\alpha(t)), g(\beta(t)))\|.$$

where  $\alpha$  and  $\beta$  range over continuous and non-decreasing reparametrizations with  $\alpha(0) = l_I$ ,  $\alpha(1) = r_I$ ,  $\beta(0) = l_J$ ,  $\beta(1) = r_J$  only.

If we drop the requirement on  $\alpha$  and  $\beta$  to be non-decreasing, we obtain a distance measure that is called the weak Fréchet distance between  $f$  and  $g$ .

A popular illustration of the Fréchet-metric is the following: Suppose a person is walking his dog, the person is walking on the one curve and the dog on the other. Both are allowed to control their speed but they are not allowed to go backwards. Then the Fréchet distance of the curves is the minimal length of a leash that is necessary for both to walk the curves from beginning to end.

## 2 Matching a Curve in a Graph

Let  $G = (V, E)$  be an undirected connected planar graph with a given embedding in  $\mathbb{R}^2$ ,  $|V| = q$ ,  $|E| =$

\*Freie Universität Berlin, Institut für Informatik, Takustr. 9, 14195 Berlin, Germany. {alt, rote}@inf.fu-berlin.de

†University of Arizona, Computer Science Dept, 1040 E 4th Street, Tucson, AZ 85721. {alon, carolaw}@cs.arizona.edu

$O(q)$ , such that  $V = \{1, \dots, q\}$  corresponds to points  $\{v_1, \dots, v_q\}$  in  $\mathbb{R}^2$ . We assume, although  $G$  is an undirected graph, that each undirected edge between vertices  $i, j \in V$  is represented by the two directed edges  $(i, j), (j, i) \in E$ . Thus  $E$  consists of directed edges, but still represents an undirected graph. Each edge  $(i, j) \in E$  is embedded as an oriented straight line segment  $s_{i,j}$  from  $v_i$  to  $v_j$ .  $s_{j,i}$  is obtained from  $s_{i,j}$  by reversing the orientation. Furthermore let  $\alpha : [0, p] \rightarrow \mathbb{R}^2$  be a polygonal curve in  $\mathbb{R}^2$ , which consists of  $p$  line segments  $\bar{\alpha}_i := \alpha|_{[i, i+1]}$  for  $i \in \{0, 1, \dots, p-1\}$ . We consider each line segment  $\bar{\alpha}_i$  to be parameterized by its *natural parametrization*, i.e.,  $\alpha(i + \lambda) = (1 - \lambda)\alpha(i) + \lambda\alpha(i + 1)$  for all  $\lambda \in [0, 1]$ . For a vertex  $i \in V$  we denote by  $\text{Adj}(i) \subseteq V$  the set of vertices adjacent to  $i$ . We identify a path  $\pi$  in  $G$  with the polygonal curve that is formed by its edges. Given  $\alpha$  and  $G$  we wish to find a path  $\pi$  in  $G$  which minimizes  $\delta_F(\alpha, \pi)$ . Note that this definition allows a path  $\pi$  in  $G$  to travel the same edges in  $G$  multiple times.

We attack this minimization problem by first solving the decision problem for which we fix  $\varepsilon > 0$  and wish to find a path (if it exists) in  $G$  such that the Fréchet distance is at most  $\varepsilon$ . Afterwards we apply parametric search, similar as in [1], to eventually solve the minimization problem. As a subproblem we consider the variant of only deciding whether there exists a path in  $G$  with the desired properties. The algorithm for the decision problem then can be used to design one for the computation of such a path.

**2.1 Basic Concepts and Algorithm Outline** If not stated otherwise let  $\varepsilon > 0$  be given. We employ the notion of the *free space*  $F_\varepsilon$  and the free space diagram  $FD_\varepsilon$  of two curves, which was introduced in [1]:

**DEFINITION 2.1.** ([1]) *Let  $f : I \rightarrow \mathbb{R}^2, g : J \rightarrow \mathbb{R}^2$  be two curves;  $I, J \subseteq \mathbb{R}$ . The set  $F_\varepsilon(f, g) := \{(s, t) \in I \times J \mid \|f(s) - g(t)\| \leq \varepsilon\}$  denotes the free space of  $f$  and  $g$ . We call the partition of  $I \times J$  into regions belonging or not belonging to  $F_\varepsilon(f, g)$  the free space diagram  $FD_\varepsilon(f, g)$ .*

We call points in  $F_\varepsilon$  *white* or *feasible* and points in  $FD_\varepsilon \setminus F_\varepsilon$  *black* or *infeasible*. See Figure 1 for an illustration.

In [1] it has been shown that  $\delta_F(f, g) \leq \varepsilon$  if and only if there exists a curve within  $F_\varepsilon(f, g)$  from  $(0, 0)$  to  $(1, 1)$  which is monotone in both coordinates. We call a curve within  $F_\varepsilon(f, g)$  *feasible*. We thus concentrate on finding a monotone feasible path in certain free space diagrams. Figure 1 shows polygonal curves  $f, g$ , a distance  $\varepsilon$ , and the corresponding free space diagram with the free space  $F_\varepsilon$ . Observe that the monotone

curve in  $F_\varepsilon(f, g)$  from  $(0, 0)$  to  $(1, 1)$  as a continuous mapping from  $[0, 1]$  to  $[0, 1]^2$  directly gives continuous increasing reparametrizations  $\alpha$  and  $\beta$ .

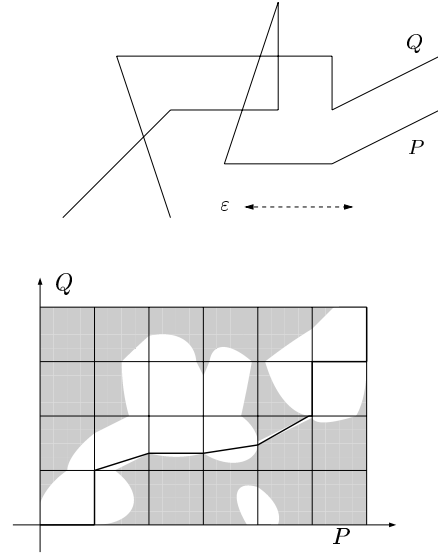


Figure 1:  
Free space diagram for two polygonal curves  $P$  and  $Q$ .  
This illustration is taken from [1].

Let  $s_{i,j}$  for all  $(i, j) \in E$  be continuously parameterized by values in  $[0, 1]$ , thus  $s_{i,j} : [0, 1] \rightarrow \mathbb{R}^2$ . For every edge  $(i, j) \in E$  consider the free space  $F_{i,j} := F_\varepsilon(\alpha, s_{i,j}) \subseteq [0, p] \times [0, 1]$ . The free space diagram  $FD_{i,j} := FD_\varepsilon(\alpha, s_{i,j})$  is the subdivision of  $[0, p] \times [0, 1]$  into the *white* points of  $F_{i,j}$  and into the *black* points of  $[0, p] \times [0, 1] \setminus F_{i,j}$ .

As shown in [1],  $FD_{i,j}$  consists of a row of  $p$  cells. Each such cell corresponds to a line segment of  $\alpha$ , and the free space in each cell is the intersection of an ellipse with that cell. For a vertex  $j \in V$  let  $FD_j := FD_\varepsilon(\alpha, v_j)$ , which is a one-dimensional free space diagram consisting of at most  $2p+1$  black or white intervals. Let  $F_j := F_\varepsilon(\alpha, v_j)$  be the corresponding one-dimensional free space, which consists of a collection of white intervals. Furthermore, let  $\mathbf{L}_j$  be the left endpoint and  $\mathbf{R}_j$  be the right endpoint of  $FD_j$ .

For each  $i \in V$  the free space diagrams  $FD_{i,j}$  and  $FD_{j,i}$  for all  $j \in \text{Adj}(i)$  have the one-dimensional free space diagram  $FD_i$  in common - as the bottom of  $FD_{i,j}$  and as the top of  $FD_{j,i}$ . Thus we can imagine to glue together the two-dimensional free space diagrams at the one-dimensional free space they have in common, according to the adjacency information of  $G$ . Like this we obtain a rather complex topological structure which we call the *free space surface* for  $G$  and  $\alpha$ . see Figure 2

for an example.

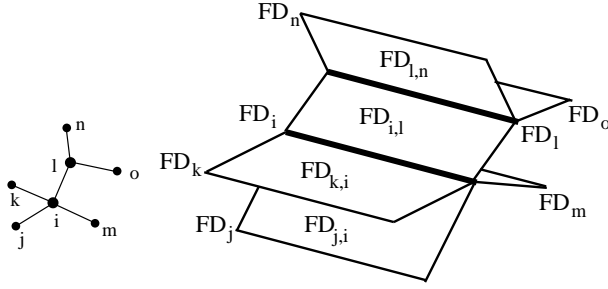


Figure 2: Example of a free space surface: Free space diagrams glued together according to the adjacency information of  $G$ .

The algorithm in [1] computes a monotone feasible path in the free space diagram of two polygonal curves in a dynamic programming fashion. We apply a related approach to our more general setting: We search for a feasible path in the free space surface. This path has to start at some white left corner  $\mathbf{L}_k$  and has to end at some white right corner  $\mathbf{R}_j$ , for two vertices  $j, k \in V$ , since the corresponding path  $\pi$  in  $G$  has to start and end in a vertex of  $G$ . Any path  $\pi$  in  $G$  selects a sequence of free space diagrams in the free space surface, whose concatenation yields  $FD_\varepsilon(\alpha, \pi)$ . Thus let us consider the following reachability information:

For every vertex  $j \in V$  let  $\mathcal{R}(j)$  be the set of all points  $u \in F_j$  for which there exists a  $k \in V$  and a path  $\pi$  from  $k$  to  $j$  in  $G$  such that there is a monotone feasible path from  $\mathbf{L}_k$  to  $u$  in  $F_\varepsilon(\alpha, \pi)$ . We call points in  $\mathcal{R}(j)$  *reachable*. We call an interval of points in  $\mathcal{R}(j)$  *reachable* if every point in it is reachable. We thus know that there is a path  $\pi$  in  $G$  with  $\delta_F(\alpha, \pi) \leq \varepsilon$  iff there is a vertex  $j \in V$  such that  $\mathbf{R}_j \in \mathcal{R}(j)$ .

Similar to [1] we first decide whether there exists a feasible path in the free space surface by computing  $\mathcal{R}(j)$  for all  $j \in V$  in a dynamic programming manner. In fact we will not store the whole  $\mathcal{R}(j)$  but only parts of it which allow us to arrive at the correct decision. The algorithm solving the decision problem consists of three stages: The *preprocessing stage* which computes the free space diagrams  $FD_{i,j}$  together with some additional reachability information, the *dynamic programming stage* which decides if there exists a feasible path in the free space surface, and the *path reconstruction stage* which constructs the path  $\pi$  in  $G$  along with feasible reparametrizations of  $\pi$  and  $\alpha$  that witness the fact that  $\delta_F(\alpha, \pi) \leq \varepsilon$ . In Subsection 2.6 we show how to apply parametric search to solve the minimization problem.

In the following we make use of a property of  $FD_{i,j}$  for each  $(i, j) \in E$ , which we call the *simplicity property* of  $FD_{i,j}$ : Each  $FD_{i,j}$  is a row of cells, and each white region in such a cell is the intersection of an ellipse with the cell boundary. Thus there is no vertical line at any position in  $FD_{i,j}$  which contains white, black, and white points alternatingly. Or in other words, the white points on a vertical line always form an interval. From this we obtain the following insight:

LEMMA 2.1. *Let  $(i, j) \in E$ , and  $u \in F_i, v \in F_j$  be white points with  $u \leq v$  for which exists a feasible monotone path in  $FD_{i,j}$  from  $u$  to  $v$ . Then for every  $u' \in F_i$  and  $v' \in F_j, u \leq u' \leq v' \leq v$  there exists a feasible monotone path in  $FD_{i,j}$  from  $u'$  to  $v'$ .*

*Proof.* Consider the feasible monotone path from  $u$  to  $v$ . Then due to the simplicity property of  $FD_{i,j}$  it is possible to go straight upward from  $u'$  until hitting this path, and similarly to go straight downward from  $v'$  until hitting this path, and stay inside the free space all the time. Stitching those pieces of paths together we obtain the desired feasible monotone path in  $FD_{i,j}$  from  $u'$  to  $v'$ .

**2.2 Preprocessing** We compute all one-dimensional free space diagrams  $FD_i$  for all  $i \in V$ . Conceptually we continue to consider the  $FD_{i,j}$  for all  $(i, j) \in E$ , but we do not need to compute them explicitly, for we capture the reachability information in the additional pointers we will compute. Let  $(i, j) \in E$  be fixed, then

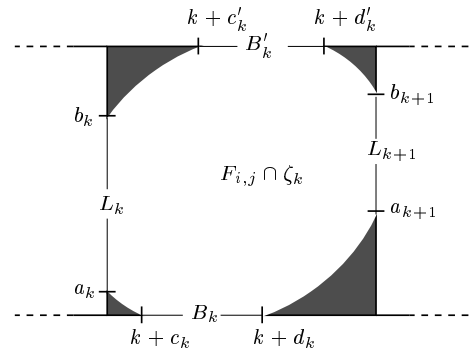


Figure 3: Intervals of the free space on the boundary of a cell.

$FD_{i,j} \subseteq [0, p] \times [0, 1]$  consists of  $p$  cells, one for each segment in  $\alpha$ . Let  $\zeta_k$  be the cell in  $FD_{i,j}$  corresponding to the  $k$ th segment  $\bar{\alpha}_k$  of  $\alpha$ ,  $0 \leq k \leq p - 1$ . Let  $L_k = [a_k, b_k]$  be the white interval on the left boundary of  $\zeta_k$ ,  $B_k = [k + c_k, k + d_k]$  be the white interval on the bottom boundary of  $\zeta_k$ ,  $B'_k = [k + c'_k, k + d'_k]$  be the white interval on the top boundary of  $\zeta_k$ . See Figure 3

for an illustration. If  $L_k$  is black then we set  $a_k := 1$  and  $b_k := 0$ . Similarly for a black  $B_k$  we set  $c_k := 1$  and  $d_k := 0$ , and for a black  $B'_k$  we set  $c'_k := 1$  and  $d'_k := 0$ . Note that the left boundary of  $\zeta_k$  is part of the vertical line segment  $\{k\} \times [0, 1]$  with respect to the free space diagram  $FD_{i,j}$ . We call  $\{k\} \times \mathbb{R}$  the *vertical line at  $k$* . We call the black parts in  $\zeta_k$ , of which there are at most four, *spikes*. In particular we call the spikes bounded from above by  $a_k$  or  $a_{k+1}$  *lower spikes*, and the spikes bounded from below by  $b_k$  or  $b_{k+1}$  *upper spikes*. We call  $a_k, a_{k+1}, b_k, b_{k+1}$  the *heights* of the corresponding spikes. Similarly, we call  $c_k, c'_k, d_k, d'_k$  *widths* of *left* and *right* spikes. We call  $k$  the *index* of the two spikes bounding  $L_k$ . Note that the interval endpoints correspond to heights or widths of spikes.

For all  $(i, j) \in E$  we compute for each white interval  $I$  of  $FD_i$  the leftmost point  $l_{i,j}(I)$  (*left pointer* or *l-pointer*) on  $FD_j$  and the rightmost point  $r_{i,j}(I)$  (*right pointer* or *r-pointer*) on  $FD_j$  which can be reached from some point in  $I$  by a monotone feasible path in  $FD_{i,j}$ . This can be done in linear time for all intervals on  $FD_j$ , see Lemma 2.3. Note that  $l_{i,j}(I)$  either equals the left endpoint of  $I$  or equals  $c'_k$  for some  $0 \leq k \leq p-1$ . For the right pointer holds  $r_{i,j}(I) = d'_k$  for some other  $0 \leq k \leq p-1$ . Note that similar reachability pointers have been used in [1] for attacking the case of closed curves. Let us call  $l(I)$  the left endpoint of  $I$ , and  $r(I)$  the right endpoint of  $I$ .

For notation purposes we identify in the following a white interval  $I$  on  $FD_i$  with a  $B_k$  for a  $0 \leq k \leq p-1$ . If a white interval on  $FD_i$  spans several cells we consider it to be composed of one white interval per cell.

For each white interval  $I$  of  $FD_i$  we store the left pointers and right pointers in two arrays that are indexed by the  $j \in \text{Adj}(i)$ . Thus each white interval  $I$  on  $FD_i$  has  $|\text{Adj}(i)|$  *l*-pointers and *r*-pointers attached to it.

The following lemma gives a characterization when points on  $FD_j$  can be reached from points on  $FD_i$  by a monotone feasible path in  $FD_{i,j}$ .

LEMMA 2.2. *Let  $(i, j) \in E$  be fixed. Let  $0 \leq k < k' \leq p-1$ , and assume that  $B_k, B'_{k'} \neq \emptyset$ . Then there is a monotone feasible path in  $FD_{i,j}$  from a point on  $B_k$  to a point on  $B'_{k'}$  if and only if*

$$\max_{i=k+1}^l a_i \leq \min_{i=l}^{k'} b_i \quad \text{for all } k < l \leq k'.$$

*Proof.* Assume there is a monotone path  $\pi$  in  $F_{i,j}$  from a point on  $B_k$  to a point on  $B'_{k'}$ . For each  $k < l \leq k'$  consider where  $\pi$  passes the vertical line at  $l$ .  $\pi$  has to pass above all  $a_i$  for  $i = k+1, \dots, l$  and below

all  $b_j$  for  $j = l, \dots, k'$ , otherwise it would not be a monotone feasible path. For the other direction, assume that (2.1) holds for all  $k < l \leq k'$ . Let  $a_{i_1}, \dots, a_{i_m}$  be the sequence of different indices that form the partial maxima of the sequence  $a_1, \dots, a_{p-1}$ , when considering its prefixes obtained by reading it from left to right. We construct  $\pi$  to start in an arbitrary point on  $B_k$ , go vertically upwards until the height  $a_{i_1}$ , go horizontally until we hit the lower spike in  $i_1$ , then visit the points  $a_{i_1}, \dots, a_{i_m}$ , and then pass horizontally until it ends under some point on  $B'_{k'}$ , which it then connects to by going vertically straight up. Two points  $a_{i_\nu}$  and  $a_{i_{\nu+1}}$  are connected in  $\pi$  by a path that starts horizontally at height  $a_{i_\nu}$  until it hits the lower spike in  $i_{\nu+1}$ . It then follows the boundary of this spike (which is monotonically increasing) until the height  $a_{i_{\nu+1}}$ . By construction  $\pi$  is monotone. Since (2.1) holds for  $l = 1, i_1, \dots, i_m$  each described piece in the path is indeed feasible.

LEMMA 2.3. *Let  $(i, j) \in E$ . Then all pointers  $l_{i,j}(B_k)$  and  $r_{i,j}(B_k)$  for all white intervals  $B_k$  on  $FD_i$ ,  $1 \leq k \leq p-1$ , can be computed in  $O(p)$  time.*

*Proof.* The left pointers  $l_{i,j}(B_k)$  for all  $0 \leq k \leq p-1$  are easily computed by a scan for increasing  $k = 0, \dots, p-1$ : Let  $k$  be fixed. If  $c_k \leq d'_k$  then we set  $l_{i,j}(B_k) := k + \max(c_k, c'_k)$ . Otherwise we greedily search for the first cell  $\zeta_{k'}$ ,  $k' > k$ , which contains a white point on its upper boundary, and such that (2.1) holds. If such a cell does not exist then  $B_k$  is black. Otherwise we set  $l_{i,j}(B_k) := k' + c'_{k'}$ . For the next iteration, i.e., for  $k$  increased by one, we only have to consider cells to the right of  $\zeta_{k'}$ , such that in total we visit every cell at most once.

The computation of the right pointers is slightly more complicated. We proceed incrementally for  $k = 0, \dots, p-1$  as follows. For each  $k$ , if  $B_k \neq \emptyset$ , we compute the largest value  $k'$  for which (2.1) holds. In order to do this we maintain a stack  $\mathcal{S} := \{i_1, \dots, i_m\}$  of indices  $k < i_1 < i_2 < \dots < i_m \leq k'$  which are the indices of those lower spikes that are horizontally visible from the vertical line at  $k'$ . In other words,  $\mathcal{S}$  is the sequence of different indices that form the partial maxima of the sequence  $a_{k+1}, \dots, a_{k'}$ , when reading it from left to right. Thus each index  $i_s \in \mathcal{S}$  is characterized by the property that  $a_{i_s} > a_l$  for all  $i_s < l \leq k'$ . We call  $\mathcal{S}$  the *partial maxima stack*, with *top* element  $i_m$ , and *bottom* element  $i_1$ . Note that for  $\mathcal{S} = \{i_1, i_2, \dots, i_m\}$  we have  $i_1 < i_2 < \dots < i_m$  and  $a_{i_1} > a_{i_2} > \dots > a_{i_m}$ . See Figure 4 for an illustration. The significance of these values is as follows: Let  $i_s < i_{s+1} \in \mathcal{S}$  be two successive indices, and let  $i_s < i \leq i_{s+1}$ . Then the lowest point on the vertical line at  $k'$  that can be reached from  $B_i$  (if

$B_i \neq \emptyset$ ) by a monotone feasible path in  $FD_{i,j}$  is  $a_{i_s+1}$ .

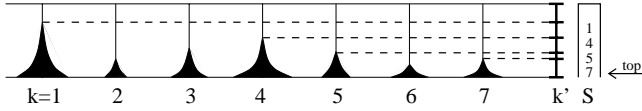


Figure 4: An example of lower spikes and their partial maxima stack  $\mathcal{S}$ .

We initialize  $\mathcal{S} = \{0\}$  and  $k' = 1$ . Let  $k = 0, \dots, p-1$  be the current value of the iteration. We maintain the invariant that (2.1) holds for the current values of  $k$  and  $k'$  throughout the algorithm. This is trivially true for the initialization case. And if we know that (2.1) holds for  $k-1$  and  $k'$ , then it immediately holds for  $k$  and  $k'$ . For fixed  $k$  we now search for the maximal  $k'$  that fulfills (2.1). (We always denote the top element of  $\mathcal{S}$  by  $a_{i_1}$  and the bottom element by  $a_{i_m}$ , although the indices and the value of  $m$  change during the algorithm.) If  $a_{i_1} > b_{k'+1}$ , then  $k'+1$  violates (2.1), thus  $k'$  is the maximal value we searched for. If  $a_{i_1} \leq b_{k'+1}$ , then we have  $\max\{a_{i_1}, a_{k'+1}\} = \max_{i=k+1}^{k'+1} a_i \leq b_{k'+1}$ , thus (2.1) holds for  $k'+1$  and we can safely increase  $k'$  by one. Now we have to maintain  $\mathcal{S}$  to represent the partial maxima of lower spikes between  $k$  and the increased value  $k'$ . For this we pop the topmost values from  $\mathcal{S}$  until  $a_{i_m} > a_{k'}$ . Finally we push  $k'$  on top. Then we start with a new iteration on  $k'$ .

Once we have found the maximal  $k'$  that fulfills (2.1), we know that there is no monotone feasible path in  $FD_{i,j}$  from any point on  $B_k$  (assuming that  $B_k \neq \emptyset$ ) to  $B_{k'+1}$ . Thus the rightmost point on  $FD_j$  that can be reached by a monotone feasible path from  $B_k$  is the first  $d'_w$  which bounds a white interval on  $FD_j$  to the left of the vertical line at  $k'+1$ .

In order to obtain all  $d'_w$  efficiently during the run of the algorithm we store  $O(p)$  *shortcut pointers* for each  $FD_i$ : At every  $k$ -th cell boundary of  $FD_i$ , for integer  $0 \leq k \leq p-1$ , we store a pointer to the rightmost white point on  $FD_i$  that lies to the left of  $k$ . If there is no such white point we set the shortcut pointer to NIL. We construct this pointer structure on the fly by computing a pointer value from the shortcut pointer to its left. Now we find  $d'_w$  by greedily searching for the next white point on  $FD_j$  to the left of  $k'+1$ . If possible we follow the next shortcut pointer; otherwise we greedily search for the first white point and compute the shortcut pointers on the way until we either hit an already computed shortcut pointer or the end of  $FD_j$ . If  $k < w$  then we set  $r_{i,j}(B_k) := w + d'_w$ . If  $k > w$  then we set  $B_k$  to be black. If  $k = w$  then if  $c_k \leq d'_k$  we set  $r_{i,j}(B_k) := k + d'_k$ , otherwise we set  $B_k$  to be black.

Finally, if  $i_1 = k+1$  then we remove  $i_1$ , i.e., the bottommost element, from  $\mathcal{S}$ . Then we start the next iteration on  $k$  with its value increased by one.

For the runtime analysis, note that  $k$  and  $k'$  are always increased, and never decreased. In each such increasing step we perform only constant time operations without counting the stack operations and the location of the  $d'_w$ . Once a value is removed from the stack (either by popping from the top, or by removing from the bottom) it is never inserted in  $\mathcal{S}$  again. Thus every integer between 1 and  $p-1$  is at most once inserted in the stack or removed from the stack. With respect to the shortcut pointers we charge every cell boundary for computing its shortcut pointer. Thus the total time to compute all  $r_{i,j}(I)$  is indeed  $O(p)$ .

**2.3 Dynamic Programming** In this stage we decide if there exists a feasible monotone path in the free space surface. Note that such a path traverses a sequence of free space diagrams  $FD_{i,j}$ . We call the part of a path that traverses one such free space diagram a *segment* of the path.

Conceptually we sweep all  $FD_{i,j}$  at once with a vertical sweep line from left to right. Let  $0 \leq x \leq p$  denote the position of the sweep line. For each  $i \in V$  we will store a set  $C_i \subseteq \mathcal{R}(i) \subseteq F_i$  of white points, which we compute in a dynamic programming manner. Throughout the algorithm we maintain the following invariant:

**DEFINITION 2.2.** ( $C_i$ ) *Let  $i \in V$  and  $x$  be any current position of the sweep line. Then  $C_i$  consists of all reachable points  $u \in \mathcal{R}(i) \subseteq FD_i$ , such that  $u \geq x$ , and for which the last segment of their associated feasible monotone path crosses or ends at the sweep line.*

Thus we are able to decide if  $\mathbf{R}_i \in \mathcal{R}(i)$  by checking if  $\mathbf{R}_i \in C_i$  for an advanced enough position  $x$  of the sweep line. Let us call a sequence of consecutive white and black intervals of  $FD_i$  a *consecutive chain* of intervals. For a consecutive chain, as well as for a single interval,  $C$  let  $l(C)$  be its left and  $r(C)$  be its right endpoint. For two consecutive chains  $C' \subseteq C$  we call  $C'$  a *consecutive subchain* of  $C$ .

**LEMMA 2.4.** *Every  $C_i$ , for  $i \in V$ , is a consecutive chain, for every value of  $x$ .*

*Proof.* Let  $x$  and let  $i \in V$  be fixed. Let  $w \in C_i$  be the largest point in  $C_i$ . By definition of  $C_i$  there is a  $j \in \text{Adj}(i)$  and a white point  $u \in F_j$  with  $u \leq x \leq w$ , such that  $u$  is reachable and there exists a monotone feasible path in  $FD_{j,i}$  from  $u$  to  $w$ . For any white point  $v \in F_i$  with  $x \leq v \leq w$  there exists by Lemma 2.1

a monotone feasible path from  $u$  to  $v$  in  $FD_{j,i}$ , which makes  $v$  especially also reachable by the same path that reaches  $u$ , concatenated by the monotone feasible path from  $u$  to  $v$ . Thus  $v \in C_i$ , and  $C_i$  is a consecutive chain.

The algorithm we present is a mixture of a sweep (since we are sweeping with a sweep line), dynamic programming (on the  $C_i$  we incrementally build up), and Dijkstra's algorithm for shortest paths (since we are computing paths using a priority queue to augment the path in a similar fashion to Dijkstra). We maintain a priority queue  $Q$  of white intervals of  $FD_i$  which are known to be reachable. More precisely, for each  $i \in V$  the first white interval of  $C_i$  (if  $C_i \neq \emptyset$ ) is stored in  $Q$ . The priority of an interval is its left endpoint. The events for the sweep line, i.e., the different values of  $x$ , are the left endpoints of the intervals in  $Q$ . Every interval in  $Q$  is part of a consecutive chain to which we store a pointer with the interval. Since  $C_i = [l(C_i), r(C_i)] \cap FD_i$  we store the  $C_i$  implicitly in constant space by storing only  $l(C_i)$  and  $r(C_i)$ .

We initialize  $Q$  with all white  $\mathbf{L}_i$  (which are degenerate intervals). For all  $i \in V$  if  $\mathbf{L}_i$  is white we set  $C_i := \mathbf{L}_i$ , otherwise  $C_i := \emptyset$ . Then we process these intervals in increasing order as follows:

1. Extract and delete the leftmost interval  $I$  from  $Q$ ; if there are several intervals with the same priority pick an arbitrary one. Advance  $x$  to  $l(I)$ .
2. Let  $C_i$  be the consecutive chain that contains  $I$ . Insert the next white interval of  $C_i$  which lies to the right of  $I$ , into  $Q$ .
3. For each  $j \in \text{Adj}(i)$  update  $C_j$  to comply with the new value of  $x$ .  $[l_{i,j}(I), r_{i,j}(I)]$  defines a consecutive chain on  $FD_j$ , whose white intervals are white intervals on  $FD_j$  which have now been identified to be reachable. Thus we need to merge  $[l_{i,j}(I), r_{i,j}(I)]$  into  $C_j$ . Knowing that  $C_j$  is a consecutive chain for every value of  $x$ , we can merge both chains together by simply considering the interval endpoints. If  $l(I) > r(C_j)$  then we discard the old  $C_j$  and replace it with  $[l_{i,j}(I), r_{i,j}(I)]$ . If the left endpoint has changed then we delete the old first interval of  $C_j$  in  $Q$  and insert the new one. Assuming an appropriate implementation of the priority queue, each operation on  $Q$  takes  $O(\log p)$  time.
4. Store for each white interval  $J$  that has been newly added to  $C_j$  (or that has been enlarged) a *path pointer* to the interval  $I$  (from which it can be reached by a monotone feasible path in  $FD_{i,j}$ ).

We process all intervals in  $Q$  until we either find a  $j \in V$  such that  $\mathbf{R}_j \in C_j$ , or until  $Q$  is empty. In the latter case there is no path  $\pi$  in  $G$  with  $\delta_F(\alpha, \pi) \leq \varepsilon$ . In the first case we know that there exists such a path, and we reconstruct it using the path pointers in the second stage of the algorithm, which is described in Subsection 2.4.

**2.4 Path Reconstruction** We assume that in the dynamic programming stage we found a  $j \in V$  with  $\mathbf{R}_j \in J$ , where  $J$  is a white interval in  $C_j$  for some position  $x$  of the sweep line. In this stage we use the path pointers to construct a path  $\pi$  in  $G$  together with a feasible monotone path in  $FD_\varepsilon(\alpha, \pi)$  which witnesses the fact that  $\delta_F(\alpha, \pi) \leq \varepsilon$ .

By construction the right endpoint of  $J$  has a path pointer attached to it. We follow this path pointer to the right endpoint of an interval  $I$ , which is a suffix of an interval on  $FD_i$  for an  $i \in \text{Adj}(j)$ . We repeat following the path pointers until we end at an  $\mathbf{L}_k$ . This way we obtain a sequence of pairs  $(i, r)$  where  $i \in V$  and  $r$  is the right endpoint of the visited interval on  $FD_i$ . We call this sequence the *path sequence*. Note that it starts with  $(k, \mathbf{L}_k)$  for a  $k \in V$ . When we extract the first component of each pair, we obtain a sequence of  $i \in V$  that represents the desired path  $\pi$  in  $G$ . The corresponding feasible monotone path in  $FD_\varepsilon(\alpha, \pi)$  can be constructed in an incremental way by following the path sequence and assuring monotonicity by using again a stack of lower spikes.

## 2.5 Time Analysis

**THEOREM 2.1.** *The described algorithm decides if there is a path  $\pi$  in  $G$  such that  $\delta_F(\alpha, \pi) \leq \varepsilon$  in  $O(pq \log q)$  time and  $O(pq)$  space. If such a path  $\pi$  exists the algorithm computes  $\pi$  together with a monotone feasible monotone path in the free space surface, in  $O(pq \log q)$  time and  $O(pq)$  space.*

*Proof.* Each  $FD_i$  has complexity  $O(p)$  and can be constructed in  $O(p)$  time. Each interval  $I$  on  $FD_i$  has  $|\text{Adj}(i)|$   $l$ -pointers and  $r$ -pointers attached to it. The number of all  $l$ - and  $r$ -pointers for all  $FD_i$  sums up to  $O(p|E|) = O(pq)$ , and can by Lemma 2.3 also be constructed in this time. Thus we need  $O(pq)$  time and space for the preprocessing.

In the dynamic programming stage we insert and delete a suffix of every white interval of any  $FD_i$ ,  $i \in V$ , at most once in  $Q$ . Also the left endpoint of a white interval of any  $FD_i$  might be changed  $|\text{Adj}(i)|$  times. Each priority queue operation needs  $O(\log q)$  time, thus  $O(pq \log q)$  altogether. For each interval in  $Q$  we consider each  $j$  in the adjacency list of its consecutive chain and spend constant time to merge

consecutive chains and construct path pointers for each such  $j$ . Altogether this sums up to  $O(p|E|) = O(pq)$  time, which together with the priority queue operations is  $O(pq \log q)$  time for the whole dynamic programming stage. We store only one consecutive chain per vertex, and  $Q$  contains at most one interval per vertex, which adds up to  $O(q)$  space. Additionally we store one path pointer per interval in  $FD_i$ , thus the space complexity for the path pointers is  $O(pq)$ .

By construction of the path pointers there is no cycle in the graph of path pointers. Thus every path pointer can be contained in a monotone feasible path in the free space surface at most once. We reconstruct a feasible path using a graph traversal in  $O(pq)$  time (since there are  $O(pq)$  path pointers). Clearly the construction of  $\pi$  in  $G$  then also needs  $O(pq)$  time.



Figure 5: Screenshot of the program.

The program has been implemented in C with a graphical user interface using OpenGL. It allows to edit the graph and the curve, to solve the decision problem, to perform binary search on  $\varepsilon$ , and it visualizes the computed feasible parametrizations in a walk-through animation. See Figure 5 for a screenshot of an example input; the found path in the graph is highlighted. The decision algorithm runs remarkably fast without specific optimizations. For example, for graphs with  $q = 700$  edges and a curve of length  $q = 420$  it runs in 5 seconds, for  $q = 1170$  and  $p = 1000$  in 35 seconds, and for  $q = 1170$  and  $p = 100$  in less than 2 seconds, on a Pentium 4 processor.

**2.6 Parametric Search** In order to find the optimal  $\varepsilon$  we apply parametric search - analogously to [1] -

to the algorithm we presented to solve the decision problem. The outcome of this algorithm depends solely on the relative positions of all possible widths and heights of spikes in all free space diagrams in the free space surface. For varying  $\varepsilon$  all those values are dependent on  $\varepsilon$ , and for the parametric search an  $\varepsilon$  is critical if it makes two of these widths or heights coincide. There are  $O(pq)$  different widths or heights of spikes. As in [1] we now apply a parallel sorting algorithm on those  $O(pq)$  values which depend on  $\varepsilon$ , and generate in that way a superset of the critical values of  $\varepsilon$  we need. By utilizing Cole's trick [2] we obtain a running time of  $O(pq \log(pq) \log q)$ , at no extra storage.

**THEOREM 2.2.** *There is an algorithm that finds a path  $\pi$  in  $G$  which minimizes  $\delta_F(\alpha, \pi)$ , in  $O(pq \log(pq) \log q)$  time and  $O(pq)$  space.*

**2.7 Variants** There are several variants of the problem setting and of the basic algorithm.

A straight-forward variant is to allow a path  $\pi$  in  $G$  to start and end not only in vertices of  $G$  but also in the middle of segments  $s_{i,j}$  for edges  $(i, j) \in E$ . In fact this can be easily integrated into our algorithm by letting a path begin (or end) on any white point on the left (or right) boundary of any  $FD_{i,j}$ .

Another variant is to ask for more monotonicity in the path  $\pi$  that is found in the graph. In our current problem setting we allow a path  $\pi$  in  $G$  to travel the same edges in  $G$  multiple times. It seems to be hard to avoid these cases without increasing the runtime immensely. However we can modify our algorithm to avoid "U-turns", i.e., to forbid a path  $\pi$  in  $G$  to travel the edge  $(i, j)$  and immediately afterwards the edge  $(j, i)$ . We incorporate this feature by storing, at every reachable white interval  $I$  on  $FD_i$ , a path pointer to each reachable interval on  $FD_j$  from which  $I$  can be reached;  $j \in \text{Adj}(i)$ . Performing a depth first traversal in this graph of path pointers we can locally exclude the option to travel back the edge from which we arrived in a vertex, and thus altogether obtain the same results as before.

The last variant is a time-space-tradeoff. We sketch this variant only very briefly due to lack of space. Note that the space of  $O(pq)$  is needed only in two places, namely in the preprocessing stage to compute the  $FD_i$  together with all  $l$ -pointers,  $r$ -pointers, and also in the dynamic programming stage to store the path pointers. We save the space for the preprocessing by integrating the computation of the  $l$ -pointers and  $r$ -pointers into the algorithm, such that we compute these pointers only when we need to access them. For this we store

a stack of currently visible lower spikes at each edge  $(i, j) \in E$ . Since this stack could contain too many spikes we actually store only an equidistant sample of it. Since during the algorithm we need to recompute the missing information between two sample points, the *spacing* of this sampling is then reflected in the running time. Let  $1 \leq t \leq \sqrt{p}$  be a given tradeoff parameter. We space the spikes in the stacks at distance  $t$ . The processing of all intervals can then be done  $O(qpt)$  time, the space that is needed is only  $O(qp/t)$  and all operations on  $Q$  altogether take  $O(qp \log q)$  as before.

For reducing space for the path pointers, we will apply a standard space-saving technique for dynamic programming [4, 3] in the path reconstruction stage. Basically we break up  $\alpha$  into several smaller pieces, compute the solution for those subparts of  $\alpha$  while keeping certain path pointer information for these subparts, and then concatenating the subresults together. Altogether we obtain the result that we can decide if there is a path  $\pi$  in  $G$  such that  $\delta_F(\alpha, \pi) \leq \varepsilon$  in  $O(pq(t + \log q))$  time and  $O(qp/t)$  space. Parametric search can be applied in a similar way as before, introducing another factor of  $\log(pq)$  to the running time.

Note that the time-space tradeoff from this section together with the approach to avoid U-turns can be used to compute the Fréchet distance for two polygonal curves with the same time-space tradeoff. Thus, at the cost of a logarithmic factor in  $q$  compared to the algorithm of [1], our algorithms yield also a time-space tradeoff for computing the Fréchet distance of curves.

### 3 Graph-to-Graph Distance

In this section we generalize the Fréchet distance to pairs of *geometric graphs*, i.e., embedded, connected graphs  $H = (V_H, E_H)$  and  $G = (V_G, E_G)$  with straight edges. Observe, that if  $H$  is not a curve there is, in general, no injective continuous parameterization  $f : [0, 1] \rightarrow H$ , so that we have to drop this condition. In the person-dog paradigm we would like to define the distance from  $H$  to  $G$  as the shortest length of a leash necessary so that the dog visits each point of the edges of  $H$  while the person traverses some part of  $G$ .

More formally, identifying  $H$  and  $G$  with the points lying on their edges we will call a mapping  $f : [0, 1] \rightarrow H$  which is continuous and surjective, a *traversal* of  $H$ . A continuous (but not necessarily surjective) mapping  $g : [0, 1] \rightarrow G$  will be called a *partial traversal* of  $G$ . The *traversal distance* from  $H$  to  $G$  is defined as

$$\delta_T(H, G) = \inf_{f, g} \max_{t \in [0, 1]} \|f(t) - g(t)\|$$

where  $f$  ranges over all traversals of  $H$  and  $g$  over all partial traversals of  $G$ . Observe, that if  $H$  and  $G$  are

polygonal chains this definition corresponds to the weak Fréchet-distance, see [1]. Also observe that the traversal distance is not a generalization of the Fréchet distance between a curve and a graph as defined in Section 2. Figures 6a and 6d show examples, where the traversal distance from  $H$  to  $G$  is small, in Figures 6b and 6c it is large. Let us first consider the decision problem,

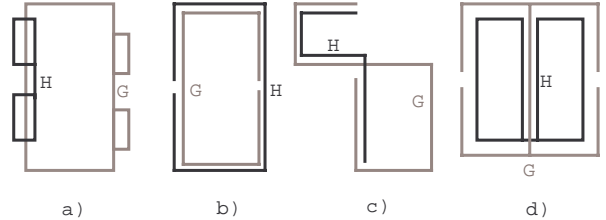


Figure 6: a), d) small b), c) large traversal distance.

i.e., determining for given  $H, G$  and  $\varepsilon \geq 0$  whether  $\delta_T(H, G) \leq \varepsilon$ . In order to find an algorithm for the decision problem, we consider for all edges  $e \in E_H$  and  $f \in E_G$  the cells  $C_{e,f}$  of the free space diagram, which can be identified with the two-dimensional unit interval  $[0, 1]^2$  within which, as was mentioned before, the freespace is obtained by the intersection with an ellipse. If  $e = (u, v)$  and  $f = (x, y)$ , we name the right, left, upper and lower sides of  $C_{e,f}$  as  $C_{v,f}$ ,  $C_{u,f}$ ,  $C_{e,y}$ , and  $C_{e,x}$ , respectively (see Figure 7). Then we identify sides with the same name, i.e. we “glue together” cells of the form  $C_{e,f}$  and  $C_{e',f}$  ( $C_{e,f}$  and  $C_{e',f}$ ) if  $f$  and  $f'$  ( $e$  and  $e'$ ) have a common endpoint  $x$  ( $u$ ) at the sides named  $C_{e,x}$  ( $C_{u,f}$ ). Thus we obtain a generalization of the free space surface from Subsection 2.1 which is a two-dimensional cell complex  $\mathcal{S}$  in three dimensions, whose facets are the are the cells  $C_{e,f}$ , whose edges are the sides  $C_{u,f}$  and  $C_{e,x}$ , and whose vertices are the points  $C_{u,x}$ , with  $e \in E_H$ ,  $f \in E_G$ ,  $u \in V_H$ ,  $x \in V_G$ . Please note that we use a slightly different notation in this section than in Section 2.

$\mathcal{S}$  contains the combined “white” freespace of all its cells and is the generalization of the freespace diagram of two curves. A continuous path on  $\mathcal{S}$  which completely lies inside the free space corresponds to a simultaneous motion on  $G$  and  $H$  keeping a distance of at most  $\varepsilon$ . Let us call these paths *feasible*.

If a feasible path  $\pi$  traverses some cell  $C_{e,f}$  then let  $I_{\pi,e,f}$  the set of those points on  $e$  that are traversed by the corresponding motion on the graphs.  $e \in E_H$  is called *satisfied* by  $\pi$  if the union

$$\bigcup_{f \in E_G} I_{\pi,e,f} = e$$

It means that all points of  $e$  are eventually traversed by the motion on the graphs corresponding to  $\pi$ . Therefore, we can conclude

LEMMA 3.1.  $\delta_T(H, G) \leq \varepsilon$ , exactly if there exists a feasible path  $\pi$  satisfying all edges  $f \in E_H$ .

In order to obtain an algorithm to test the condition of Lemma 3.1 we introduce the *traversal graph*  $\mathcal{T}$ . The vertices of  $\mathcal{T}$  are the one-dimensional facets  $C_{u,f}$  and  $C_{e,x}$  of the cell complex  $\mathcal{S}$ , with  $e \in E_H$ ,  $f \in E_G$ ,  $u \in V_H$ ,  $x \in V_G$ . Two such facets are connected by an edge of  $\mathcal{T}$  exactly if they are both incident to some cell  $C_{e,f}$  and if there is a connection between both by a curve through the free space of  $C_{e,f}$ , see Figure 7. Thus, to each edge of  $\mathcal{T}$  we can assign a cell of the free space. On the other hand, each cell is assigned to at most six edges. It follows that  $\mathcal{T}$  has  $O(pq)$  edges where  $p = |E_G|$  and  $q = |E_H|$ .

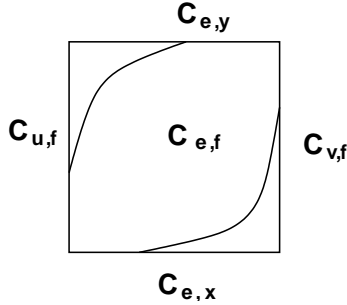


Figure 7: Edges of the traversal graph.

For  $e \in E_H, f \in E_G$  let  $J_{e,f}$  be the set of all points on  $e$  that have distance at most  $\varepsilon$  from  $f$ , i.e., the projection of the freespace in  $C_{e,f}$  to  $e$ . Any path  $\pi$  with the properties described in Lemma 3.1 yields a path in the traversal graph  $\mathcal{T}$  whose edges are assigned to the cells  $C_{e,f}$  traversed by  $\pi$ . Since any edge  $e \in E_H$  is satisfied by  $\pi$  it must be  $\bigcup_f J_{e,f} = e$  where  $f$  ranges over all cells  $C_{e,f}$  traversed by  $\pi$ . Consequently, the equation is true if  $f$  ranges over all edges in  $E_G$  such that  $C_{e,f}$  is an edge in the connected component  $\mathcal{C}$  of  $\mathcal{T}$  containing  $\pi$ . Our algorithm for the decision problem is based on the fact that also the converse is true:

LEMMA 3.2.  $\delta_T(H, G) \leq \varepsilon$ , exactly if there exists a connected component  $\mathcal{C} = (V_{\mathcal{C}}, E_{\mathcal{C}})$  of the traversal graph  $\mathcal{T}$  such that for all  $e \in E_H$

$$\bigcup_f J_{e,f} = e$$

where  $f$  ranges over all edges in  $E_G$  where  $C_{e,f}$  is assigned to an edge in  $E_{\mathcal{C}}$ .

To see the converse suppose that  $\mathcal{C}$  is a connected component of  $\mathcal{T}$  with this property. Then we construct a path  $\pi$  on  $\mathcal{S}$  as follows:  $\pi$  traverses all vertices of  $\mathcal{C}$  by, say, breadth-first-search. For each cell  $C_{e,f}$  visited,  $\pi$  makes sure that  $I_{\pi,e,f} = J_{e,f}$  by visiting the leftmost and the rightmost point of the freespace (see Figure 8).

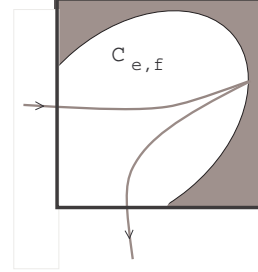


Figure 8: Motion of  $\pi$  within  $C_{e,f}$ .

Then for all  $e \in E_H$

$$\bigcup_f I_{e,f,\pi} = \bigcup_f J_{e,f}$$

where  $f$  ranges over all edges in  $E_G$  such that  $C_{e,f}$  is a cell visited by  $\pi$ . Therefore  $\pi$  satisfies all edges  $e \in E_H$  and  $\delta_T(H, G) \leq \varepsilon$  by Lemma 3.1.

Lemma 3.2 enables us to give quite a simple **algorithm** for solving the decision problem. In fact, given geometric graphs  $G$  and  $H$  and  $\varepsilon > 0$ , we first determine all freespace cells  $C_{e,f}, e \in E_H, f \in E_G$ , and the traversal graph  $\mathcal{T}$ . By breadth-first-search we determine all connected components of  $\mathcal{T}$  and we check for each of them whether the condition of Lemma 3.2 holds for each edge  $e \in E_H$ . If this is the case for at least one connected component, the algorithm answers "yes", otherwise "no".

In order to determine the runtime of this algorithm, we observe that the breadth-first-search in total visits  $O(pq)$  cells  $C_{e,f}$  since there are  $O(pq)$  edges in  $\mathcal{T}$ . For each cell we have to add the interval  $J_{e,f}$  to the portion of  $e$  covered so far which takes time  $O(\log pq)$ .

In order to solve the *optimization problem* observe that for the smallest  $\varepsilon$  for which the decision problem has a positive answer, there are two possibilities. On the one hand it could be that the left endpoint of some interval  $J_{e,f}$  equals the right endpoint of another one  $J_{e,f'}$ , so that edge  $e$  gets satisfied at that point. On the other hand it could be that the free space in some cell  $C_{e,f}$  touches one of the sides of the cell, i.e., the traversal graph  $\mathcal{T}$  changes. Therefore, in order to solve the optimization problem by performing a parametric

search using Cole's approach [2] with a fast parallel sorting algorithm for the endpoints of the intervals  $J_{e,f}$  including the values 0 and 1, to take care of the critical values of the second type. Since there are  $O(pq)$  such endpoints and the decision problem can be solved in time  $O(pq \log pq)$  we obtain an  $O(pq \log^2 pq)$  algorithm for the computation problem. We summarize

**THEOREM 3.1.** *Given two geometric graphs  $G$  and  $H$  and  $\varepsilon > 0$ , it can be decided whether  $\delta_T(H, G) \leq \varepsilon$  in time  $O(pq \log pq)$  by the algorithm given above, where  $p$  and  $q$  are the numbers of edges of  $G$  and  $H$ , respectively. The traversal distance from  $H$  to  $G$  can be computed in time  $O(pq \log^2 pq)$ .*

#### 4 Acknowledgements

We thank Scott Howard Morris for introducing us to the application of matching GPS curves, and Lingeswaran Palaniappan for implementing the algorithm.

#### References

- [1] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *Internat. J. Comput. Geom. Appl.*, 5:75–91, 1995.
- [2] R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *Journal of the ACM*, 34(1):200–208, 1987.
- [3] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997.
- [4] D.S. Hirschberg. Algorithms for the longest common subsequence problem. *J. ACM*, 24:664–675, 1977.