Realizability of Free Spaces of Curves

Hugo A. Akitaya
Department of Computer Science, University of Massachusetts Lowell, MA, USA

Maike Buchin
Department of Computer Science, Ruhr University Bochum, Germany

Majid Mirzanezhad
Transportation Research Institute, University of Michigan, Ann Arbor, MI, USA

Leonie Ryvkin
Department of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands

Carola Wenk
Department of Computer Science, Tulane University, New Orleans, LA, USA

Abstract

The free space diagram is a popular tool to compute the well-known Fréchet distance. As the Fréchet distance is used in many different fields, many variants have been established to cover the specific needs of these applications. Often the question arises whether a certain pattern in the free space diagram is *realizable*, i.e., whether there exists a pair of polygonal chains whose free space diagram corresponds to it. The answer to this question may help in deciding the computational complexity of these distance measures, as well as allowing to design more efficient algorithms for restricted input classes that avoid certain free space patterns. Therefore we study the inverse problem: Given a potential free space diagram, do there exist curves that generate this diagram?

Our problem of interest is closely tied to the classic Distance Geometry problem. We settle the complexity of Distance Geometry in $\mathbb{R}^{>2}$, showing $\exists\mathbb{R}$-hardness. We use this to show that for curves in $\mathbb{R}^{\geq 2}$ the realizability problem is $\exists\mathbb{R}$-complete, both for continuous and for discrete Fréchet distance. We prove that the continuous case in $\mathbb{R}^1$ is only weakly NP-hard, and we provide a pseudo-polynomial time algorithm and show that it is fixed-parameter tractable. Interestingly, for the discrete case in $\mathbb{R}^1$ we show that the problem becomes solvable in polynomial time.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Fréchet distance, Distance Geometry, free space diagram, inverse problem

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.3

Related Version *Full Version*: https://arxiv.org/abs/2311.07573

Funding *Carola Wenk*: partially supported by NSF grant CCF 2107434.

1 Introduction

The Fréchet distance is arguably the most popular distance measure for curves in computational geometry and has been studied extensively in the last years. It has application in various fields, including geographic data analysis and the comparison of protein chains [25, 26, 35, 28]. For the latter application, typically the well-established variant, the discrete Fréchet distance, is used. The standard tool for computing the Fréchet distance of two curves is the *free space diagram*, which is the cross-product of the parameter spaces of the curves partitioned into *free space* and its complement, where free space is the sublevel set of the distance function for a given $\varepsilon > 0$. For two piecewise linear curves of m and n line segments parameterized by their natural arc-length parametrization, it is well-known that the free space diagram consists of mn cells, and the free space in each cell has the shape of a cropped ellipse [5]. The

Fréchet distance is at most ε if and only if there exists a monotone path in the free space diagram that covers the parameter spaces of both curves. Hence, to compute the Fréchet distance one searches for such a path in the free space diagram.

For different applications, many variants of the Fréchet distance have been developed, which are typically also computed using the free space diagram. The discrete Fréchet distance relies on (a discretization of the free space diagram) the *free space matrix*: For two discretized curves given by n and m points, respectively, the $n \times m$ free space matrix with entries $a_{i,j} \in \{0, 1\}$ captures whether two points i, j of different curves lie within $\varepsilon > 0$ distance of one another, or not. The discrete Fréchet distance of two curves is at most ε iff there exists a monotone "path" of 1 entries connecting opposing corners in the free space matrix.

Runtimes of the resulting algorithms usually depend on the complexity of the free space diagram or free space matrix [23, 8, 10]. It is known that neither Fréchet distance nor discrete Fréchet distance can be computed in subquadratic time unless SETH fails [12, 16, 13]. However, there are several faster algorithms for special curve classes such as [6, 24], which exploit a special structure of the free space diagram. These complexity bounds always consider the worst-case complexity of the free space diagram, but not every free space diagram can be realized by a pair of curves. Also, some variants of the Fréchet distance have proven to be NP-hard to decide [4, 17], some of which build certain free space diagrams for the reduction.

Here we therefore study the inverse problem: Given a (potential) free space diagram (free space matrix), do there exist curves (ordered point sets) that generate this free space diagram or matrix? To our knowledge this *free space realizability problem* has so far only been studied for special cases [32, 18, 3]. Understanding it will give structural insights into free spaces and the computation of the Fréchet distance, in particular for special curve classes. Note that although we gained a good understanding of the realizability problem in the settings described below, other settings remain to be investigated further. We are particularly interested in studying settings where less information is given in the free space diagram or matrix, e.g. only some cells or only cell boundaries are provided with the input.

Overview. We give results (see Table 1) for both the continuous and the discrete variant of the problem, with free space diagram and free space matrix inputs, and curves may be realized in $\mathbb{R}^d$, with $d = 1$ or $d \geq 2$. We show that for curves in $\mathbb{R}^{\geq 2}$ the realizability problem is $\exists\mathbb{R}$-complete both for the continuous case (Section 3) and for the discrete case (Section 5). For the continuous case in $\mathbb{R}^2$, algorithms are known only for special cases [32, 18]. For curves in $\mathbb{R}^1$, the problem in the discrete case interestingly becomes solvable in polynomial time (Section 6), while in the continuous case, it is weakly NP-hard (Section 4) and fixed-parameter tractable, and we also provide a pseudo-polynomial time algorithm (Section 4).

Table 1 Overview of our and known results.

Input	$\mathbb{R}^d$	Results
Free Space Diagram	$d \geq 2$	$\exists\mathbb{R}$-complete (Theorem 4) **Section 3** Algorithms for special cases ([32, 18])
	$d = 1$	weakly NP-hard (Theorem 7) **Section 4** FPT $O(mn2^k)$ (Theorem 8) **Section 4** Pseudo-poly time (Theorem 13) **Section 4**
Free Space Matrix	$d \geq 2$	$\exists\mathbb{R}$-complete (Theorem 14) **Section 5**
	$d = 1$	$O(nm^2)$ (Theorem 16) **Section 6**

Related problems. The exploration of inverse problems is a recurrent subject in computational geometry, often applied to recognition and reconstruction problems. Notable examples are the *inverse Voronoi Diagram* problem [9] and the *visibility graph recognition* problem [11]. Our problem of interest can be viewed as a curve embeddability problem given certain proximity criteria on edge lengths and point-to-point distances between two curves in their free space diagram. Our results have a significant impact on problems involving distance constraints on geometric graphs, such as *Distance Geometry* and *Disk Intersection Graphs*.

Distance Geometry. Our results closely relate to problems involving distance constraints on geometric graphs, such as the *Distance Geometry* problem, which is a classic inverse problem in computational geometry. It involves embedding an abstract weighted graph in $\mathbb{R}^d$ Euclidean space such that the Euclidean length of each edge corresponds to its weight [33]. This problem is equivalent to *Linkage Realizability*, where the edges correspond to rigid bars in a mechanical linkage [33]. While distance geometry was shown to be NP-hard in the 70s, its membership in NP remained open until Schaefer showed $\exists\mathbb{R}$-completeness in 2012 [34] for $\mathbb{R}^2$. Whether the problem remains $\exists\mathbb{R}$-hard in higher dimensions was posed as an open problem by Schaefer. To show that the continuous version of the free space realizability problem in $\mathbb{R}^{\geq 2}$ is $\exists\mathbb{R}$-hard we reduce from distance geometry, using a gadget from [33].

Unit Sphere Graphs. The sphericity of a graph is the minimum dimension for which the graph has a unit sphere representation. Unit sphere graph realizability is known to be $\exists\mathbb{R}$-hard. Havel first introduced the study of sphericity in the context of molecular conformation [27]. A unit sphere graph is an intersection graph of unit spheres and can be seen as a complete graph where each edge is marked with a distance constraint ≤ 1 or > 1. The problem of realizing a free space matrix corresponds to realizing a complete bipartite graph where each edge is marked with a distance constraint ≤ 1 or > 1 (Section 5). This defines a class of graphs, as do unit disk graphs. In contrast to a unit disk graph, there are pairs of vertices (the ones in the same partite set) whose distances are dispensable. A similar class are visibility graphs, the recognition of which is also known to be $\exists\mathbb{R}$-complete [19].

Bipartite Distance-Constrained Graphs. Modeling distance constraints in general (non-complete) graphs is useful when data is unavailable between every pair of nodes or due to the topology of the underlying network. E.g., heteronuclear NMR is used to obtain less cluttered and less noisy data [30]. This allows the inference of distances between two different types of atoms, and thus, the distances constraints form a bipartite graph.

2 Preliminaries

Continuous case. Let $P = (p_0, \ldots, p_n)$ and $Q = (q_0, \ldots, q_m)$ be polygonal curves in $\mathbb{R}^d$ of lengths ℓ_P and ℓ_Q, continuously parameterized by arc-length, i.e. $p_i = P(i)$ and $q_j = Q(j)$ for $i \in [n]$, $j \in [m]$, where $[n] = \{1, \cdots, n\}$. Given $\varepsilon > 0$, their *free space* is defined as $F_\varepsilon(P, Q) = \{(r, t) \mid \|P(r) - Q(t)\| \leq \varepsilon\}$. The *free space diagram* puts this information in an $m \times n$ grid: We define $D_\varepsilon(P, Q)$ as the colored rectangle $R = [0, \ell_P] \times [0, \ell_Q] \subseteq \mathbb{R}^2$, where a point $(p, q) \in R$ is colored white iff $(p, q) \in F_\varepsilon(P, Q)$. The grid X, which is the set of segments $\{p_i \times [0, \ell_Q] \mid i \in \{0, \ldots, n\}\} \cup \{[0, \ell_P] \times q_j \mid j \in \{0, \ldots, m\}\}$, subdivides R into $n \times m$ cells $C_{i,j}$. We call a single cell $C_{i,j}$ *empty* if $C_{i,j} \cap F_\varepsilon = \emptyset$ and *full* if $C_{i,j} \cap F_\varepsilon = C_{i,j}$. If $\emptyset \neq C_{i,j} \cap F_\varepsilon \neq C_{i,j}$, the cell $C_{i,j}$ is called *partially full*. A diagram D_ε is called *realizable* if there exist curves P, Q such that $D_\varepsilon(P, Q) = D_\varepsilon$. We assume that the exact lengths of

all cell boundaries and the equation of each component's boundary curve are part of the input. We consider the real RAM computation model throughout the paper. In Figure 1, the leftmost diagram is not realizable: Upon fixing the placement of segments corresponding to cell $C_{1,1}$, we have two options to place the remaining segments such that cell $C_{2,2}$ is realized. This either induces components in none of the remaining cells, or in both.

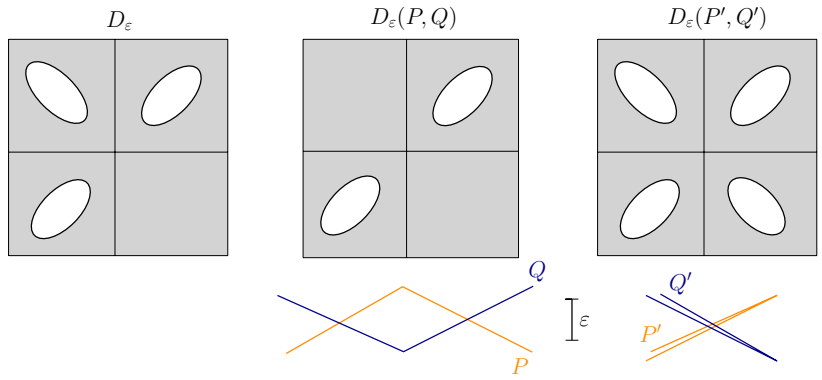

Figure 1 Given diagram D_ε, there are two ways to place curves P, Q in $\mathbb{R}^2$. Neither realizes D_ε.

In the following, we denote the ball of radius r centered at a point $x \in \mathbb{R}^d$ as $\mathcal{B}_r(x) := \{p \in \mathbb{R}^d \mid \|x - p\| \leq r\}$. The *$\varepsilon$-neighborhood* of an object X is given by $\bigcup_{x \in X} \mathcal{B}_\varepsilon(x)$. We denote by $s_i^P = \overline{p_{i-1}p_i}$, $i \in [n]$, the line segment connecting consecutive vertices of P, and define s_i^Q analogously. In $\mathbb{R}^1$, if two consecutive segments s_i^P, s_{i+1}^P have different orientations (segments are placed on top of each other), we say the curve *folds* at the common *folding vertex* p_i. Else, we say that the curve is *straight* at p_i. It is known that the free space of two lines has the shape of a cropped ellipse with axis at $\pm 45°$ [5, 31]. For curves in $\mathbb{R}^1$, the lines are necessarily parallel, hence the ellipse degenerates to a slab bounded by lines at $\pm 45°$ [14].

Partially full cells. We now establish a necessary condition for curves P, Q to realize D_ε that is used in Sections 3–4. Partially full cells give us information about the relative placement of the segments. We use the term *relative placement* of two segments to mean that we know the distances between the intersection point of the lines containing the segments and the segment endpoints. Note that after fixing the position of one segment, this still allows for two symmetric placements of the second segment.

▶ **Lemma 1.** *Given a partially full free space cell, $\varepsilon > 0$, and four points on the boundary of the ellipse in the cell, none of which are mirror images of another with respect to the ellipse's major and minor axes, we can compute the corresponding segments' relative placement.*

The proof (see Lemma 5.5 in [32]) relies on knowing that a cell is an ellipse at 45°. Note that we obtain much less information from full or empty cells, namely only that the segments do or do not lie within or not within distance ε from each other.

Definitions: Discrete case. For discrete polygonal curves (i.e., point sequences) P, Q with n and m points, resp., the free space is defined as $F_\varepsilon(P,Q) = \{(i,j) \in [n] \times [m] \mid \|p_i - q_j\| \leq \varepsilon\}$. We define the free space matrix $M_\varepsilon(P,Q)$ as the $n \times m$ matrix featuring entries $a_{i,j} \in \{0,1\}$, $i \in [n]$, $j \in [m]$ where $a_{i,j} = 1$ if and only if $(i,j) \in F_\varepsilon(P,Q)$. For a given matrix M_ε we ask whether there exist curves P, Q such that $M_\varepsilon = M_\varepsilon(P,Q)$.

3 $\exists\mathbb{R}$-Completeness for Continuous Curves

We first show that given a diagram D_ε, the problem of finding two curves in $\mathbb{R}^2$ that realize D_ε is $\exists\mathbb{R}$-complete. We then generalize to higher dimensions in Section 3.1. Containment in $\exists\mathbb{R}$ is shown by expressing the problem using real inequalities, see Lemma 17, Appendix A.

We reduce from the problem of deciding whether a linkage has a planar realization which was shown $\exists\mathbb{R}$-hard by Abel et al. [1, 2]. A *mechanical linkage* is a mechanism made of rigid bars connected at hinges. The input is a weighted graph $G = (V(G), E(G), \ell_G)$, where ℓ_G is the weight function, and a function $\Pi\colon W \to \mathbb{R}^2$, where $W \subseteq V(G)$, that represents vertices whose positions are *pinned*. A *configuration* C of a linkage $\mathcal{L} = (G, \Pi)$ is a straight-line drawing of G where the length of each edge $e \in E(G)$ is $\ell_G(e)$ and the position of each vertex $w \in W$ is $\Pi(w)$. The linkage realization problem asks whether a given linkage admits a configuration. A configuration C is *noncrossing* if C is a plane drawing. Abel et al. [1, 2] showed that the linkage realization problem remains hard for a series of restrictions on the input linkage $\mathcal{L}$. We restate a direct consequence of Theorems 2.2.13 and 2.4.6 in [2]. Although not all conditions in the theorem below are explicitly stated in [2, Theorem 2.2.13], they can be directly inferred by their construction in [2, Section 2.7].

▶ **Theorem 2** (Simplified from [2], Theorems 2.2.13 and 2.4.6)**.** *Given a linkage $\mathcal{L} = (G, \Pi)$ and a combinatorial embedding (clockwise circular order of edges around each vertex) σ of G, deciding whether there exists a planar realization of $\mathcal{L}$ is $\exists\mathbb{R}$-hard even if the following constraints are enforced:*

1. *G is connected, and the length of every edge is an integer.*
2. *A set of edge disjoint subgraphs H of G can be assigned* rigid*, i.e., each angle between consecutive incident edges in H is prescribed from $\{90°, 180°, 270°, 360°\}$. Each subgraph H is a tree, and an edge in $E(G) \setminus E(H)$ incident to H must be incident to a leaf of H.*
3. *Only three vertices are pinned ($|\Pi| = 3$), all three belong to the same rigid subgraph H (described in constraint (2)), and they are not collinear.*
4. *For every noncrossing configuration C of $\mathcal{L}$ that satisfies constraints (1–3), holds:*
   a. *C agrees with σ.*
   b. *Angles that are not prescribed by constraint (2) lie strictly between $60°$ and $240°$.*
   c. *The minimum distance of a vertex and a nonincident edge is at least a constant ϕ.*

We call a vertex *rigid* if it is incident to at least two edges of the same rigid subgraph H, and *nonrigid* otherwise. By (2), every angle incident to a rigid vertex is prescribed while no angle in a nonrigid vertex is prescribed (which by (4b) can only vary in the interval $(60°, 240°)$). Note that distance geometry is equivalent to linkage realization with $\Pi = \emptyset$. Since (3) makes Π irrelevant, Theorem 2 also implies hardness for distance geometry.

Reduction description. Given $\mathcal{L}$ and σ satisfying the constraints in Theorem 2, we construct an instance D_ε as follows. A full example can be seen in Figure 2. The idea is to build a free space such that realizing curves trace out the linkage, following the given combinatorial embedding. For this, we first transform G into a tree. The angle constraints in the linkage can also be enforced in the free space using a specific gadget.

While there is a cycle in G, split one edge in a cycle by placing a new vertex in its midpoint and performing a vertex split, creating two copies of the new vertex, each attached to half of the original edge. We end up with a tree T. Let T' be the multigraph obtained by doubling each edge of T. Intuitively, D_ε forces the curves P and Q to roughly trace a planar Eulerian circuit of T' using the combinatorial embedding σ. (Up to a reflection and

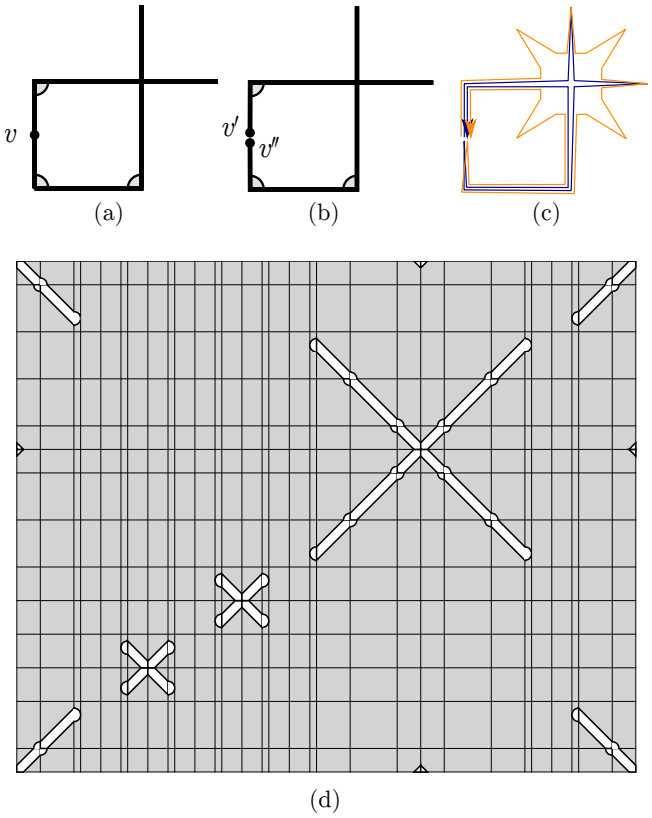

Figure 2 Example of our reduction from linkage realizability to free space diagram realizability. (a) An input linkage $\mathcal{L} = (G, \Pi)$ and a subdivision vertex v in a cycle of G. Rigid vertices are marked with gray angles. (b) Splitting v transforms G into a tree T. (c) The curves P and Q in $\mathbb{R}^2$ obtained from T. (d) The obtained free space diagram.

translation since D_ε can only specify the relative placement of P and Q.) Q is exactly a planar Eulerian circuit of T' while P traces the same circuit but avoids an ε-neighborhood of each nonrigid vertex using our *angle gadget* (described later), which allows these angles to lie freely between 60° and 240°. Both P and Q trace the "outline" σ *counterclockwise*. W.l.o.g. assume $\phi \geq 6$, scaling the linkage by a constant factor if necessary. We chose $\varepsilon = 1$ so that edges of P and Q that correspond to an edge e of G are close to each other and far from other edges. Since every partially full cell determines the relative position of the corresponding pair of edges, the four edges (two from P and two from Q) that correspond to the traversal of e are fixed relative to one another and lie on top of each other. They then simulate edge e. The angle gadget guarantees flexibility so that the angle between incident edges can vary accordingly. We add free space components to make the newly introduced subdivision vertices rigid: Their relative position is locked by Lemma 1 forming a 180° angle, see the four small components on the sides of the free space diagram in Figure 2d.

The *angle gadget*, see Figure 3, is represented by the 12 free space cells shown in Figure 3(b). It is located at a small neighborhood of a vertex v of Q; the figure only shows the portion of the free space relative to this neighborhood. Note that v is a degree-2 copy of a vertex v^* of G. For clarity, we refer to all the copies of v^* in Q with different labels (by construction, there are $\deg(v^*)$ copies of each $v^* \in V(G)$, except for the starting vertex of the Eulerian circuit, which has an extra copy). Let $\overrightarrow{e_1}$ and $\overrightarrow{e_2}$ be the two edges of Q incident to v, and

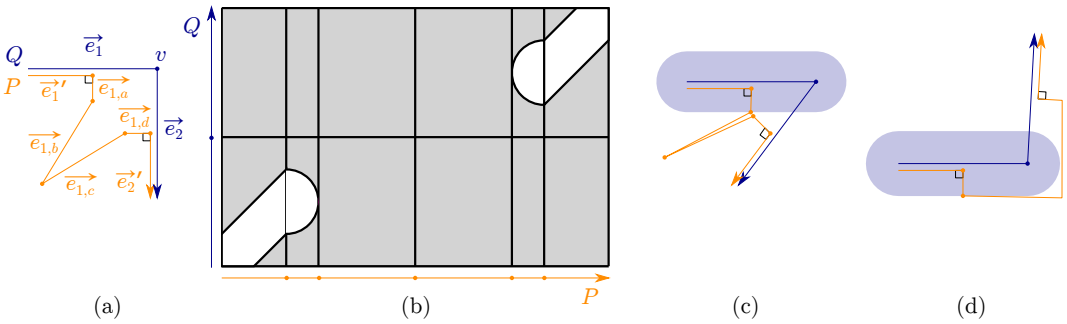

Figure 3 The angle gadget. (a) The 90° configuration and (b) its free space diagram. (c) and (d) show the extremal configurations of the gadget with angles $2 \cdot \tan^{-1}(1/2) \approx 53.13°$ and 270°, resp.

let $\overleftarrow{e_1}$ and $\overleftarrow{e_2}$ be the corresponding copies going in the opposite direction in Q, respectively. Locally, P has two edges $\overrightarrow{e_1}'$ and $\overrightarrow{e_2}'$ that overlap with $\overrightarrow{e_1}$ and $\overrightarrow{e_2}$, respectively. We enforce the overlap by making all free space cells relative to $\overrightarrow{e_1}'$ (resp., $\overrightarrow{e_2}'$) empty except for the ones relative to $\overrightarrow{e_1}$ and $\overleftarrow{e_1}$ (resp., $\overrightarrow{e_2}$ and $\overleftarrow{e_2}$) which are partially full, containing an upward and downward 45° full strip. The distance between v and the endpoints of $\overrightarrow{e_1}'$ and $\overrightarrow{e_2}'$ closest to v is 2 by Lemma 1. We place four edges $(\overrightarrow{e_{1,a}}, \overrightarrow{e_{1,b}}, \overrightarrow{e_{1,c}}, \overrightarrow{e_{1,d}})$ between $\overrightarrow{e_1}'$ and $\overrightarrow{e_2}'$ of lengths 1, 3, 3, and 1 in this order. Only edges of length 1 have corresponding partially full cells: $C_{\overrightarrow{e_{1,a}},\overrightarrow{e_1}}$ and $C_{\overrightarrow{e_{1,a}},\overleftarrow{e_1}}$ contain half of a disk of radius 1.

▶ **Lemma 3.** *Given a realization of P and Q, assume that $(\overrightarrow{e_{1,a}}, \overrightarrow{e_{1,b}}, \overrightarrow{e_{1,c}}, \overrightarrow{e_{1,d}})$ lie to the right of $(\overrightarrow{e_1}, \overrightarrow{e_2})$. Then, $\overrightarrow{e_1}$ and $\overleftarrow{e_1}$ (resp., $\overrightarrow{e_2}$ and $\overleftarrow{e_2}$) lie exactly on top of each other, and the angle to the right of $(\overrightarrow{e_1}, \overrightarrow{e_2})$ is strictly between $2 \cdot \tan^{-1}(1/2) \approx 53.13°$ and 270°.*

Proof. The fact that $\overrightarrow{e_1}$ and $\overleftarrow{e_1}$ lie exactly on top of each other is a consequence of applying Lemma 1 to $\overrightarrow{e_1}$ and $\overrightarrow{e_1}'$, and to $\overrightarrow{e_1}'$ and $\overleftarrow{e_1}$. We now focus on the angle constraint. Note that by Lemma 1, the relative positions of $\overrightarrow{e_1}$ and $\overrightarrow{e_{1,a}}$ (resp., $\overrightarrow{e_2}$ and $\overrightarrow{e_{1,d}}$) is fixed. If we fix the positions of $\overrightarrow{e_{1,a}}$ and $\overrightarrow{e_{1,d}}$, then the positions of $\overrightarrow{e_{1,b}}$ and $\overrightarrow{e_{1,c}}$ are completely determined: There are two points whose distance is 3 from the endpoints of $\overrightarrow{e_{1,a}}$ and $\overrightarrow{e_{1,d}}$; one of them causes $\overrightarrow{e_{1,b}}$ and $\overrightarrow{e_{1,c}}$ to intersect with Q which cannot happen since their free space cells are empty. If the angle is $2 \cdot \tan^{-1}(1/2)$ or smaller, the common endpoint of $\overrightarrow{e_{1,c}}$ and $\overrightarrow{e_{1,d}}$ would lie in the closed ε-neighborhood of $\overrightarrow{e_1}$ and $C_{\overrightarrow{e_1},\overrightarrow{e_{1,c}}}$ would not be empty (Figure 3(c)), a contradiction. If the angle is 270° or greater, a portion of $\overrightarrow{e_{1,b}}$ would lie in the closed ε-neighborhood of $\overrightarrow{e_1}$ and $C_{\overrightarrow{e_1},\overrightarrow{e_{1,b}}}$ would not be empty (Figure 3(d)), a contradiction. For all values in between there is a placement for $\overrightarrow{e_{1,b}}$ and $\overrightarrow{e_{1,c}}$ away from $\overrightarrow{e_1}$ and $\overrightarrow{e_2}$, making the section of the free space diagram exactly as required. ◀

Using Lemma 3 we can simulate a linkage $\mathcal{L}$ subject to the constraints in Theorem 2 using curves given by D_ε, obtaining the following theorem.

▶ **Lemma 4.** *It is $\exists\mathbb{R}$-complete to decide if a given free space diagram is realizable in $\mathbb{R}^2$.*

Proof. The described reduction produces a free space diagram D_ε with of size $O(|E(G)|^2)$: Q has length $2|E(G)|$ and each edge in $|E(G)|$ generates up to 10 segments in P, depending on whether the endpoints are rigid or not. The runtime is linear in the size of D_ε: each row corresponding to an edge of P has precisely two partially full cells. All other cells are empty.

Given a positive instance of linkage realization, Theorem 2(4) and Lemma 3 guarantee that we can find a placement of P and Q realizing D_ε as described in the reduction. The other direction is a little more subtle. D_ε forces Q to trace σ exactly: using Lemma 1 with

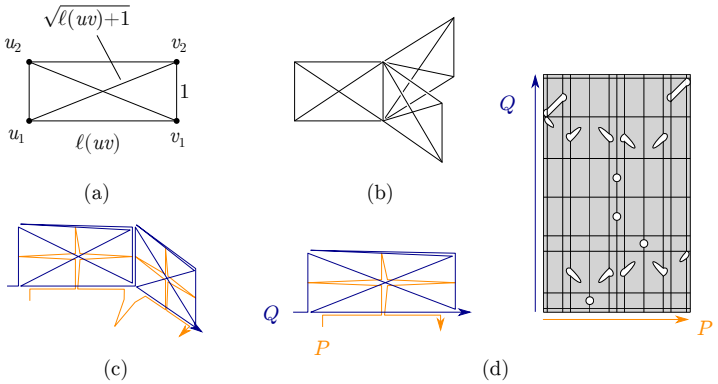

Figure 4 (a) The dimension gadget corresponding to an edge uv with length $\ell(uv)$. (b) Three gadgets in $\mathbb{R}^3$ corresponding to a degree-3 vertex. The gadget forces all vertices to be in one of two planes. (c) The realization of an angle gadget after applying the dimension gadget. (d) The free space and its realization (perturbed for clarity).

transitivity constraints the two edges of Q corresponding to an edge in $E(G)$ to lie exactly on top of each other, while the angle gadgets force the circular order around each vertex. By Lemmas 3 and 18, Q traces a noncrossing configuration of $\mathcal{L}$ exactly. If there is a valid placement of P and Q one can find a noncrossing configuration of $\mathcal{L}$ obtained by the image of Q. If such a configuration does not satisfy Theorem 2(4), that would contradict Theorem 2. Thus the promise in Theorem 2(4) must also be fulfilled by the Fréchet realization instance and the angles in each angle gadget would indeed be between 60° and 240°. ◀

3.1 Higher Dimensions

In order to show that free space realization is $\exists\mathbb{R}$-hard in higher dimensions, we show that the realizability of linkages and, thus, distance geometry are also $\exists\mathbb{R}$-hard. Here, the linkage realization is not required to be injective since we are in $\mathbb{R}^{>2}$, but the reduction will force crossings to only happen between predictable pairs of edges. This will be important in our reduction to free space realization since crossings between the curves appear in the free space diagram. We remark that, although all the ingredients of this proof were already known, the claim does not appear in the literature to the best of the authors' knowledge.

▶ **Theorem 5.** *Linkage Realization and Distance Geometry are $\exists\mathbb{R}$-hard in $\mathbb{R}^{\geq 2}$.*

Proof. Recall that linkage realization with no pinned vertices is equivalent to distance geometry. The main ingredient of this proof is the *dimension gadget* shown in Figure 4 that appears in [33]. The gadget is isomorphic to K_4 which is *globally rigid* in $\mathbb{R}^2$ [20], meaning that there is a unique embedding of the gadget in $\mathbb{R}^2$, and every realization of the gadget in $\mathbb{R}^{>2}$ is congruent with the planar realization. Given a linkage $\mathcal{L}$ satisfying the constraints of Theorem 2, replace every edge of G by a copy of the dimension gadget. Note that every vertex v is now represented by two vertices v_1 and v_2. We call the resulting linkage $\mathcal{L}'$. The gadgets force all vertices v_1 for all $v \in V(G)$ to be in the same $(k-1)$-hyperplane, perpendicular to the edges v_1v_2. Thus, $\mathcal{L}'$ is realizable in $\mathbb{R}^d$ iff $\mathcal{L}$ is realizable in $\mathbb{R}^{(d-1)}$. ◀

▶ **Theorem 6.** *It is $\exists\mathbb{R}$-complete to decide if a given free space diagram is realizable in $\mathbb{R}^{\geq 2}$.*

Proof. We adapt the dimension gadget to the free space diagram realizability problem. We first argue for $\mathbb{R}^3$. Figure 4(d) shows the adapted gadget and its free space. We use the same reduction as in Lemma 4, but replacing the edges of Q and the edges of P that overlap edges

of Q with the modified dimension gadget as follows. Each edge of Q is replaced by a path of length 7, and each edge of P that lies in the interior of an edge of Q is replaced by a path of length 10. The free space diagram between the two paths force the path of Q to be embedded as the dimension gadget. Two-dimension gadgets are neighbors if their corresponding edges share an endpoint. The cells between these paths and other non-neighbor dimension gadgets are empty. The two length-ε edges of P in the angle gadget ($\overrightarrow{e_{1,a}}$ and $\overrightarrow{e_{1,d}}$) induce partially full cells in the free space of dimension gadget (first and last collumns in the free space diagram of Figure 4(d)), forcing these edges to be perpendicular to the plane of the dimension gadget. (See Figure 4(c)) The cells of the two length-3ε edges of P in the angle gadget ($\overrightarrow{e_{1,b}}$ and $\overrightarrow{e_{1,c}}$) remain empty and thus they must be realized far from the dimension gadget. Note that the ε-neighborhood of v_1 and v_2 does not intersect P, and that the ε-neighborhood of the edges of P in the angle gadget still does not intersect Q leaving the dihedral angle between the planes of the neighbor dimension gadgets to vary as in Lemma 3. Thus the embedding of P and Q corresponds to a realization of $\mathcal{L}'$.

For dimension $d > 3$, we recursively apply the dimension gadget construction in the following way. Note that in the $d-1$-dimensional construction each edge of Q overlaps with at least one edge of P (possibly degenerate to a vertex). We recursively replace each edge of Q with the dimension gadget construction as normal. We split one edge of P that overlaps with the edge of Q at its midpoint and insert the length-8 path forming a cross that contains the midpoints of the edges in the dimension gadget of Q as in Figure 4(d). ◀

4 NP-Hardness and Algorithmic Results for Continuous Curves in $\mathbb{R}^1$

We briefly consider the realizability problem for curves in $\mathbb{R}^1$. In this case, the curves have less space to be placed in and hence the free space diagram has limited "configurations". Cells are still empty, full or partially full cells, but now free space ellipses degenerate to slabs, and the white space is bounded by parallel line segments oriented at $+$ or $-45°$, see [15, 31]. Here, we present only sketches. For details, we refer to [3]. We note that for curves in 1D the problem is weakly NP-hard by a reduction from the Partition problem. The reduction is similar to the hardness of ruler-folding.

▶ **Theorem 7.** *Realizability of continuous curves in $\mathbb{R}^1$ is weakly NP-complete.*

Next, we sketch an FPT-algorithm for continuous curves in $\mathbb{R}^1$. Inspired by computational origami [22], we observe that a given diagram D_ε is realizable iff it can be folded at the grid lines so that the white space is aligned (overlapping only with other white space) into a single convex component. In [3], we developed an algorithm to enumerate and check the different foldings, inspired by the algorithm for *simple-foldability in* $\mathbb{R}^1$ [7]. Our algorithm runs in exponential time $O(mn2^k)$, where k is the total number of (vertical and horizontal) grid lines of D_ε that do not intersect the white space (completely gray or completely white).

▶ **Theorem 8.** *Given an $m \times n$ diagram D_ε, in $O(mn2^k)$ time one can find curves P and Q in $\mathbb{R}^1$, if they exist, such that $D_\varepsilon = D_\varepsilon(P, Q)$.*

We now describe an algorithm whose input is a diagram D_ε where the dimensions of each cell are integers upper-bounded by W, and that outputs a pair of curves P, Q in $\mathbb{R}^1$ such that $D_\varepsilon = D_\varepsilon(P, Q)$ if they exist. Otherwise, it returns `false`. We use the limited placement options of curves in $\mathbb{R}^1$; the main technical ingredient is the use of dynamic programming to decide the placement of the portions of P and Q for which we have no explicit information.

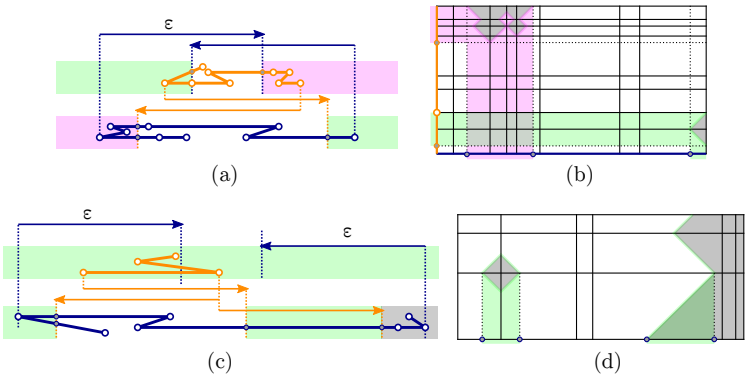

Figure 5 Regions defined by curves P (orange) and Q (blue). Certainty regions are green or pink, subdivision vertices are gray, and uncertainty regions are white (middle) or gray (left/right).

We use the following placement graph[1] G: The vertices $V(G)$ are the set of segments in the bipartite graph between segments of P and Q where an edge (v_i^P, v_j^Q) encodes that the cell $C_{i,j}$ is partially full. G can be computed in $\mathcal{O}(mn)$ time. By Lemma 3.1 in [3], if G has a single component, we can either compute P and Q, or report that no such curves exist in $\mathbb{R}^1$ in $\mathcal{O}(mn)$ time. We now show a key property of G for curves in $\mathbb{R}^1$. We say that a curve in $\mathbb{R}^1$ *spans* a distance w if its image in $\mathbb{R}^1$ is an interval of length w. A component of G is a *singleton* component if its size is one (i.e., a single vertex).

▶ **Lemma 9.** *If P and Q are two curves in $\mathbb{R}^1$, then the placement graph G computed from $D_\varepsilon(P,Q)$ has at most two non-singleton components. If either P or Q spans more than 2ε, then G has at most one non-singleton component.*

Proof. Let p_ℓ and p_r be the leftmost and rightmost points of P, respectively. We first prove the claim when P, without loss of generality, spans more than 2ε. For contradiction assume there are 2 non-singleton components in G. Every point of Q in $[p_\ell - \varepsilon, p_r + \varepsilon]$ has exactly distance ε to some point in P and thus defines the boundary of a free-space component and can be assigned an edge of G. By continuity, every maximal subcurve of Q in $[p_\ell - \varepsilon, p_r + \varepsilon]$ corresponds to edges in the same component of G. By transitivity, any two overlapping subcurves of Q in $[p_\ell - \varepsilon, p_r + \varepsilon]$ are also represented in the same component of G as both have at least one point at distance exactly ε from the same point in P. Thus the two components in G correspond to nonoverlapping maximal subcurves of Q in $[p_\ell - \varepsilon, p_r + \varepsilon]$ and there is no third subcurve of Q that overlaps the first two. Then, Q is disconnected, a contradiction.

Now, consider the case that both P and Q span less than 2ε. The points in $\mathbb{R}^1$ that are exactly at distance ε from some point in P form the intervals $[p_\ell - \varepsilon, p_r - \varepsilon]$ and $[p_\ell + \varepsilon, p_r + \varepsilon]$. The same argument as above shows that the subcurves of Q in each of these intervals define a single component in G. Thus, there are at most 2 non-singleton components. ◀

Algorithm description. First, we subdivide the two curves based on the orthogonal projections of the free space's boundary (see Figure 5). We introduce subdivision vertices so that each point in the interior of a segment is either:

1. farther than ε from any point in the other curve (an edge whose corresponding row or column in D_ε is completely empty);

[1] This is a variation of the placement graph used for the same problem for continuous curves in $\mathbb{R}^2$ [32].

2. within ε distance from every point in the other curve (an edge whose corresponding row or column in D_ε is completely full); or
3. at exactly ε distance from a point of the other curve (an edge covered by the orthogonal projection of the boundary of the free space).

We partition the segments of both curves based on these three types. Singleton components of G correspond to segments of types (1) and (2), and vertices of non-singleton components correspond to segments of type (3). The first correspond to segments of one curve that are farther than ε from every point in the other curve. The presence of type (2) segments implies that one of the curves spans less than 2ε. Adding subdivision vertices does not asymptotically increase the complexity of the problem as each segment is subdivided at most twice.

For each of the two curves, we partition $\mathbb{R}$ into up to 5 regions used to embed the segments of each of the types based on containment in the ε-neighborhoods of the extreme points of the other curve. Let p_ℓ and p_r be the leftmost and rightmost points of P. We note that we do not have previous knowledge of the points p_ℓ and p_r but we later describe how to infer information about these points from D_ε. The regions serve as an abstraction that allows us to divide the problem into subproblems. If the balls $\mathcal{B}_\varepsilon(p_\ell)$ and $\mathcal{B}_\varepsilon(p_r)$ intersect, we call the interval $[p_r - \varepsilon, p_\ell + \varepsilon]$ the *middle uncertainty region* (colored white in Figure 5(a) and (c)). Segments of type (2) must be embedded in this region. The intervals of $\mathbb{R}^1$ contained in a single disk are called *left* and *right certainty regions*, respectively $[p_\ell - \varepsilon, p_r - \varepsilon]$ and $[p_\ell + \varepsilon, p_r + \varepsilon]$ (colored green or pink in Figure 5(a) and in the bottom curve in (c)). If $\mathcal{B}_\varepsilon(p_\ell)$ and $\mathcal{B}_\varepsilon(p_r)$ do not intersect, then we call the interval $[p_\ell - \varepsilon, p_r + \varepsilon]$ the *middle certainty region* (colored green in the top curve of Figure 5(c)). The segments of type (3) must be embedded in these regions. In both cases, we call the intervals $(-\infty, p_\ell - \varepsilon]$ and $[p_r + \varepsilon, \infty)$ the *left* and *right uncertainty regions* (colored gray in Figure 5. Note that the figure only shows a closeup view and the only visible portion of a right uncertainty region is shown for the bottom curve in (c)). The segments of type (1) must be embedded in these regions.

▶ **Lemma 10.** *Given D_ε, in $O(nm)$ time we can partition the segments of Q into the three types and assign each segment to a region.*

Proof. The orthogonal projection of the components can be computed by a traversal of the free space's boundary. Thus, we can compute the subdivision vertices in $O(nm)$ time. The types of all segments can be inferred from D_ε in $O(nm)$ time. We can compute G in $O(nm)$ time. If there are two non-singleton components, we arbitrarily fix the orientation of one segment and use Lemma 3.1 in [3] to decide the relative placement for their respective segments. Note that the type of the segments adjacent to a segment in a certainty region determines which of the components is the left and which is the right certainty region: the left certainty region is adjacent to segments of type (2) on its right boundary. ◀

We can use Lemma 3.1 in [3] to determine the relative embedding of P and Q in certainty regions. It remains to determine whether the subcurves in uncertainty regions can be embedded. We use a dynamic program (DP) to solve the problem in each uncertainty region separately. We further divide the problem into two cases depending on whether we know the relative position of the boundaries of the respective region. The input of each DP is a maximal subcurve of P (resp., Q) in an uncertainty region. The DP computes the possible placements of the subcurve for a set of boundary constraints. We later describe how to combine the output of all the DPs into a single solution.

Fixed boundary subproblem. If one of the curves does not have edges of type (2), by Lemma 3.1 in [3] we know the size of the uncertainty regions. We define DP problems for each maximal subcurve in an uncertainty region whose value is `true` iff it is possible to

realize the subcurve in the region. We define subproblems based on a suffix of the subcurve and the coordinate of the first point of the suffix (details are left to a full version of this paper). The recursive definition tries embedding the next edge oriented towards the right or left, thus each subproblem depends on only two subproblems. The number of subproblems depends on the number of segments in the subcurve and the size of the uncertainty region.

▶ **Lemma 11.** *One can compute each fixed boundary subproblem defined by a subcurve Q' with n' segments, each with an integer length of at most W, and an integer interval $[0, r]$, in $O(n' \cdot \min(r, n'W))$ time.*

Proof. Since $k \in \{1, \ldots, n'\}$ and $s \in \{0, \ldots, r\}$ there are at most $O(n'r)$ subproblems. We can also upper-bound s by $n'W$ since this is the maximum length of the image of Q' (which is necessary in the case when $r = \infty$). Each subproblem can be computed in $O(1)$ time. Thus the total runtime is $O(n' \cdot \min(r, n'W))$. ◀

Variable boundary subproblem. When both curves have segments of type (2), the size of the middle uncertainty region of one curve depends on the size of the middle uncertainty region of the other. We similarly define a DP problem for each maximal subcurve in an uncertainty region. However, each subproblem is also defined by a suffix of the subcurve, the coordinate of the first point, and, additionally, the size of the uncertainty region. In this case, the size of the uncertainty region is upper-bounded by 2ε.

▶ **Lemma 12.** *For variable boundary subproblems, in $O(\max(n, m) \cdot \varepsilon^2)$ time, one can compute all DP tables and, if there exist P and Q in $\mathbb{R}^1$ that realize D_ε, find r_P and r_Q that are compatible with a solution to all subproblems, where r_P and r_Q denote the sizes of the middle uncertainty regions of P and Q respectively.*

Proof. There are $O(m + n)$ DP tables since this is the upper bound on the number of maximal subcurves in the middle uncertainty regions. Each table has $O(n' \cdot \varepsilon^2)$ subproblems, each can be computed in $O(1)$ time, where n' is the size of the maximal subcurve. Thus, it takes $O(\max(n, m) \cdot \varepsilon^2)$ to compute all DP tables. For each table, we can keep track in a separate data structure what values of α have an entry $R(i, ., \alpha) = \texttt{true}$. Then, given values for r_P and r_Q, we can check whether there exist a compatible solution in each table in $O(1)$ time per table. Thus, we can try all possible $r_P, r_Q \in \{1, \ldots, 2\varepsilon\}$ searching for values compatible with a solution for each DP problem. Then, searching for a set of compatible solutions takes $O(\max(n, m) \cdot \varepsilon^2)$ time. ◀

▶ **Theorem 13.** *Given an $m \times n$ free space diagram D_ε, where $n \geq m$, every cell has integer dimensions of at most W, and ε is an integer, we can produce two curves in $\mathbb{R}^1$ that realize D_ε or answer `false` if no such curves exist in time $O(\max(n\varepsilon^2, n^2W))$.*

5 ∃ℝ-Completeness for Discrete Curves in $\mathbb{R}^2$

We now turn to the discrete Fréchet distance, and prove that realizability by curves in $\mathbb{R}^{\geq 2}$ for a given free space matrix is also ∃ℝ-complete. We reduce from d-Stretchability, which asks whether there exists an arrangement of hyperplanes in $\mathbb{R}^{\geq 2}$ that realizes a combinatorial description. The formal description follows in the next paragraph. We use the machinery developed by Kang and Müller [29] to show that recognizing a d-sphere graph (generalization of unit disk graphs in $\mathbb{R}^d$) is ∃ℝ-hard for $d \geq 2$. (Although only NP-hardness is claimed in [29], their proof also extends to ∃ℝ-hardness as noted in their conclusion.) Recall that we explain in Section 1 that, though they are similar, our problem differs from d-sphericity.

An instance of d-Stretchability is given by a set $S \subseteq \{-,+\}^n$ of size $1+\binom{n+1}{2}$. An arrangement of n hyperplanes divides $\mathbb{R}^d$ into $1+\binom{n+1}{2}$ cells. Each vector in S corresponds to a cell in a potential arrangement. We denote by $\mathbf{v}_j \in S$ the jth vector in S and by $\mathbf{v}_j[i]$ its ith coordinate. Then $\mathbf{v}_j[i] = -$ (resp., $\mathbf{v}_j[i] = +$) if the corresponding cell is below (resp., above) the ith hyperplane. Note that $(-,\dots,-)$ and $(+,\dots,+)$ must be in S, and so we assume they are respectively $\mathbf{v}_1$ and $\mathbf{v}_2$. The problem asks whether S is the combinatorial description of an arrangement of n hyperplanes.

Reduction. Given an instance S of d-Stretchability of n hyperplanes we construct a $2n \times |S|$ free space matrix M_ε as follows. We partition P (whose vertices correspond to rows of M_ε) into two subcurves $P^+ = (a_1,\dots,a_n)$ and $P^- = (b_1,\dots,b_n)$. Informally, a_i (resp., b_i) will be a point in the upper (resp., lower) halfspace of a hyperplane ℓ_i in the arrangement. Each column j of M_ε (i.e., vertex of Q) represents a vector in $\mathbf{v}_j \in S$, that is, $M_\varepsilon[i][j] = 0$ and $M_\varepsilon[n+i][j] = 1$ (resp., $M_\varepsilon[i][j] = 1$ and $M_\varepsilon[n+i][j] = 0$) if $\mathbf{v}_j[i] = -$ (resp., $\mathbf{v}_j[i] = +$).

▶ **Theorem 14.** *Given a free space matrix M_ε, it is $\exists\mathbb{R}$-complete to decide whether there exists a pair of curves P and Q in $\mathbb{R}^{\geq 2}$ that realizes M_ε.*

Proof. Containment in $\exists\mathbb{R}$ can be proven by a straightforward reduction to $\exists\mathbb{R}$ similar to the proof of Lemma 17. We now focus on the reduction defined above. It is clear that it runs in polynomial time. Assume that there exists a pair of curves P and Q that realizes M_ε. Refer to Figure 6(a). We use the labels of point of P defined in the reduction and assume $Q = (q_1,\dots,q_{|S|})$. Recall that, informally, points q_1 and q_2 represent vectors $\mathbf{v}_1 = (-,\dots,-)$ and $\mathbf{v}_2 = (+,\dots,+)$, respectively. Rotate the solution in order to make the vector $\overrightarrow{q_1q_2}$ vertical and pointing upwards. We build a hyperplane arrangement as follows. For each $i \in [n]$, create a hyperplane ℓ_i bisecting the segment a_ib_i. Now, we argue that q_j, $j \in \{1,\dots,|S|\}$ is in a cell in the produced arrangement with description $\mathbf{v}_j$. Let C_1 and C_2 be the cells in the arrangements of circles of radius ε containing q_1 and q_2, respectively. By definition, if $\mathbf{v}_j[i] = +$, then q_j must be within ε distance from a_i and farther than ε from b_i, that is $q_j \in \mathcal{B}_\varepsilon(a_i) \setminus \mathcal{B}_\varepsilon(b_i)$. Thus, $C_1 = (\bigcap_{i=1}^n \mathcal{B}_\varepsilon(b_i) \setminus \bigcup_{i=1}^n \mathcal{B}_\varepsilon(a_i))$ and $C_2 = (\bigcap_{i=1}^n \mathcal{B}_\varepsilon(a_i) \setminus \bigcup_{i=1}^n \mathcal{B}_\varepsilon(b_i))$. Note that every hyperplane ℓ_i must separate C_1 and C_2 by definition. Thus every ℓ_i intersects the line segment $\overline{q_1q_2}$. We focus on a specific hyperplane ℓ_i. Without loss of generality assume $\mathbf{v}_j[i] = +$. Then, $\mathcal{B}_\varepsilon(a_i) \setminus \mathcal{B}_\varepsilon(b_i)$ is above ℓ_i and so is q_j. Therefore, the produced hyperplane arrangement realizes S.

Now assume that there exists a hyperplane arrangement realizing S. Refer to Figure 6(b). For each cell in the arrangement described by $\mathbf{v}_j$, choose a point q_j in the interior of the cell. As before, every hyperplane intersects the line segment $\overline{q_1q_2}$, since q_1 is below all the hyperplanes and q_2 is above. Let t_i be the intersection of ℓ_i and $\overline{q_1q_2}$. Define the balls $\mathcal{B}_r(w^+_{i,r})$ and $\mathcal{B}_r(w^-_{i,r})$ respectively above and below ℓ_i, tangent to ℓ_i at t_i. Note that $\mathcal{B}_r(w^+_{i,r})$ (resp., $\mathcal{B}_r(w^-_{i,r})$) equals the upper (resp., lower) halfspace of ℓ_i when $r \to \infty$. Thus, for sufficiently large r, $\mathcal{B}_r(w^+_{i,r})$ contains all points q_j above ℓ_i and $\mathcal{B}_r(w^-_{i,r})$ contains all points q_j below ℓ_i. Let r^* be a sufficiently large r such that the previous statement is true for all $i \in [n]$. Scale the entire construction to make $r^* = \varepsilon$. Then, we can construct P by making $a_i = w^+_{i,\varepsilon}$ and $b_i = w^-_{i,\varepsilon}$. Now, each q_j is contained in the appropriate cell of the arrangement of circles of radius ε centered at points of P. Thus, the constructed P and Q realize M_ε. ◀

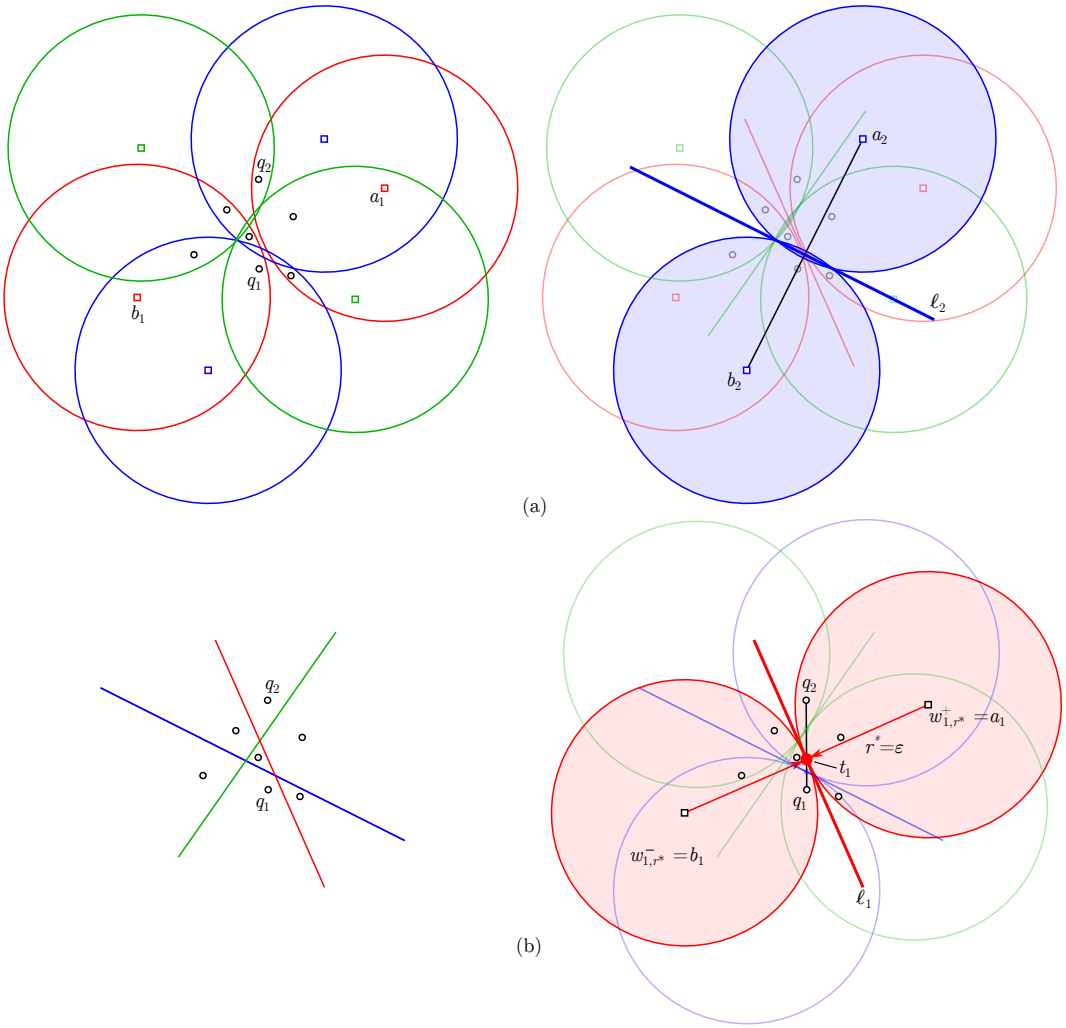

Figure 6 Reduction from $S = \{(-,-,-),(+,+,+),(-,-,+),(-,+,+),(-,+,-),(+,+-),(+,-,-)\}$. (a) Transforming a solution to M_ε into a line arrangement that realizes S. (b) Transforming a solution to S into curves P and Q that realize M_ε. Here squares represent points of P and circles represent points of Q.

6 A Polynomial Time Algorithm for Discrete Curves in $\mathbb{R}^1$

We now turn to realizability for discrete curves in $\mathbb{R}^1$ and show that this can be decided in polynomial time. We lend our main idea from the *unit-interval graph recognition* (UIGR) in [21]: given an abstract graph G whose nodes are intervals and two intervals intersect iff there is an edge between the two corresponding nodes in G, the goal is to find a placement of intervals in the real line such that the intersections induced by them fulfill G. See Figure 7 for an example. In free space realizability, we derive a unit interval graph G from the free space matrix M_ε. We then adapt the idea in [21] to handle the realizability in our case.

First, we borrow some notations from [21]. For a vertex $v \in G$ the neighboring vertices of v is denoted by $N(v)$. We also define $N[v] = N(v) \cup \{v\}$. Two vertices u and v are *indistinguishable* if $N[u] = N[v]$. In order to recognize a unit-interval graph, Corneil et al. [21] propose a linear-time algorithm: (i) find the left anchor in G (the left-most interval in

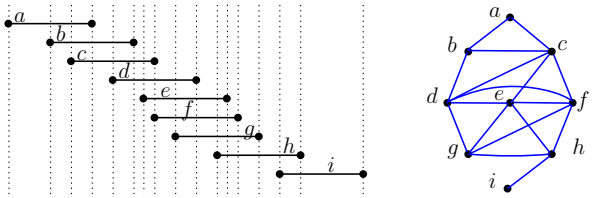

Figure 7 An abstract graph of intervals and its recognition in $\mathbb{R}^1$.

the recognition), (2) perform a BFS search starting at the left anchor to get a *partial order* of the intervals, meaning that some groups of intervals are ordered properly, but still intervals belonging to each group need to be re-ordered, and (3) refine the partial order, i.e., the intervals within in each group, to get the *global order*. If the global order exists, return "YES", and "NO", otherwise. To handle (1), they perform a BFS from an arbitrary node in G. Find a vertex z at the last level L_t of the BFS search such that $\deg(z) = \min\{\deg(w) : w \in L_t\}$. In (2), they locally order the vertices in G based upon the level in which they are encountered along the BFS search from z. Finally in (3), in each level L_k obtained in (2) they sort each vertex $v \in L_k$ in an increasing order of $D(v) = |\mathsf{Next}(N(v))| - |\mathsf{Prev}(N(v))|$. Here, $\mathsf{Next}(v)$ is the set of adjacent vertices to v in L_{k+1} and $\mathsf{Prev}(v)$ is the set of adjacent vertices to v in L_{k-1}. Next, for each vertex v in G, they compute $\alpha(v) = \min\{\text{order}(u) : \ u \in N[v]\}$ and $\omega(v) = \max\{\text{order}(u) : \ u \in N[v]\}$, where $\text{order}(u)$ is the order in which u is encountered on the line. In the end, if there exists a v for which $\alpha(v) \neq \omega(v)$, "NO" is returned, and "YES" is returned, otherwise. We modify Step (3) to handle our case.

The problem of realizing M_ε is more restrictive than UIGR as follows. Let $\varepsilon = 1/2$. Similar to the $\mathbb{R}^2$ case, we can see the realization of $Q = (q_1, \ldots, q_n)$ as an arrangement of intervals $\mathcal{B}_\varepsilon(q_j)$ partitioning $\mathbb{R}^1$. Each row i of M_ε describes a "cell" in the arrangement (which is an interval in $\mathbb{R}^1$) where we place a point p_i of P. Thus, M_ε requires a unit interval realization of P with not only prescribed adjacencies in G but prescribed cells. Our goal is to take the partial order that we get from (2), and refine it to be closer to a global order using information from M_ε. In the following, we refer to vertices of G and the interval they refer to interchangeably, and we call the maximal set of vertices that are pairwise indistinguishable an *equivalence class*. See Appendix B for a full algorithmic description.
DiscreteRealizabilityAlgo(M_ε) : **(1)** Construct the unit-interval graph G from M_ε. **(2)** Choose a left anchor v_0 by running a BFS search on G. **(3)** Perform a BFS on G starting at v_0 to obtain a partial order of the intervals. **(4)** Refine the partial order by sorting the intervals at each level of the BFS under the criterion of $D(v) = |\mathsf{Next}(N(v))| - |\mathsf{Prev}(N(v))|$. **(5)** Refine the partial order D to an order D': For each row of M_ε, place intervals of entry 1 in the equivalence class C towards those intervals that don't belong to C whose entries are 1 and orders are different than intervals in C. **(6)** Extend the partial order defined by the BFS levels and D' to a global order breaking ties arbitrarily. **(7)** Verify whether the produced arrangement is compatible with M_ε or not.

▶ **Lemma 15.** *The algorithm returns "YES" if and only if M_ε is realizable.*

Proof. By Step (7), it is clear that when the algorithm returns "YES" the instance M_ε is realizable. If the algorithm returns "NO", we show that there are no P and Q realizing M_ε. Note that the constraints in the ordering obtained from Steps (1–4) are the same as for UIGR. Thus, we must show that if G is not a unit interval graph, then M_ε is not realizable. We show the constrapositive: if M_ε is realizable, then G is a unit interval graph. As discussed before, the realization of Q implies the realization of a unit interval graph G^* whose intervals

are $\mathcal{B}_\varepsilon(q_j)$, for $\varepsilon = 1/2$. It is clear that G^* is a supergraph of G since the required cells in the interval arrangement exist containing points of P. Let q_iq_j be an edge that exists in G^* and not in G such that q_i is to the left of q_j and with shortest interval $\mathcal{B}_\varepsilon(q_i) \cap \mathcal{B}_\varepsilon(q_j)$. Since q_iq_j is not in G, $\mathcal{B}_\varepsilon(q_i) \cap \mathcal{B}_\varepsilon(q_j)$ contain no points of P and all the cells in this interval are not necessary. By the assumption that $\mathcal{B}_\varepsilon(q_i) \cap \mathcal{B}_\varepsilon(q_j)$ is shortest, there is no point q_k of Q to the right of q_j whose interval $\mathcal{B}_\varepsilon(q_k)$ intersects $\mathcal{B}_\varepsilon(q_i) \cap \mathcal{B}_\varepsilon(q_j)$. Let S be the set of points including q_j and all points q_k of Q to the right of q_j whose interval $\mathcal{B}_\varepsilon(q_k)$ does not intersect $\mathcal{B}_\varepsilon(q_j)$. Move all points in S until the intersection $\mathcal{B}_\varepsilon(q_i) \cap \mathcal{B}_\varepsilon(q_j)$ disappears. By construction, ass cells previously in $\mathcal{B}_\varepsilon(q_i) \cap \mathcal{B}_\varepsilon(q_j)$ disappear. We show that no other cell does, and thus the modified Q is still a solution for M_ε. A cell to the left of $q_j - \varepsilon$ is not affected since we only move points to the left of q_j. A cell to the right of $q_j + \varepsilon$ would disappear if $\mathcal{B}_\varepsilon(q_j)$ starts to intersect with an interval that it didn't before. This does not happen since S contains all such intervals. A cell in $\mathcal{B}_\varepsilon(q_j) \setminus \mathcal{B}_\varepsilon(q_i)$ would disappear if an interval defined by $q_k \in S \setminus \{q_j\}$ stopped intersecting another interval. Recall that there are no intervals whose right endpoint are between $q_j + \varepsilon$ and $q_j + \varepsilon + |\mathcal{B}_\varepsilon(q_i) \cap \mathcal{B}_\varepsilon(q_j)|$ by the "shortest" assumption. Then, such cells do not exist. Thus, we produced a solution to M_ε whose interval intersection graph has fewer edges than G^*. Applying this argument successively we conclude that there exist a solution whose intersection graph is G.

We now show that Step (5) is necessary. Suppose there are four intervals $\{a, b, c, d\}$ in row r, $I_r = \{a, c\}$, $C = \{b, c, d\}$, and $C' = \{c\}$. Also by assumption $a <_D b =_D c =_D d$. For the sake of contradiction, suppose that there exists a positive solution that does not place the interval c before $C \setminus C' = \{b, d\}$ and after $I_r \setminus C = \{a\}$. Placing c before a implies that $c <_D a$ which is a contradiction. Placing c after b implies that the intersection $\mathcal{B}_\varepsilon(a) \cap \mathcal{B}_\varepsilon(c)$ is contained in $\mathcal{B}_\varepsilon(b)$. This means that the cell required by $I_r = \{a, c\}$ does not exist, which is again a contradiction.

Finally, we analyse Steps (6–7). The only worry is that there might exist two extensions of the partial order produced in Step (5) such that one is a positive solution and the other isn't. Let q_i and q_j be two incomparable vertices in the partial order. Since Step (5) refines equivalence classes, q_i and q_j are also in the same equivalence class C. Then, there exist no row containing an interval not in C whose entries relative to q_i and q_j are different. If there is a row containing only a proper subset of C, the arrangement must contain a cell in which the proper subset of C intersect and that does not intersect any other interval. Since intervals in C intersect the same intervals by definition, including intervals that must be to the left and to the right of intervals in C (note that we add v_0 and v_f so that every equivalence class has a predecessor and successor), this cell cannot exist. Then Step (7) returns "NO". Else, the columns relative to q_i and q_j are identical and interchangeable. ◀

The construction of G takes $O(k^2n)$ time where $k \leq m$ is the maximum number of entries filled with 1 over all rows in M_ε, since G might contain n cliques of size k. The other steps can be implemented in linear time. The full algorithm description is given in Appendix B.

▶ **Theorem 16.** *Given an $n \times m$ free space matrix M_ε, we can decide whether there exist curves P and Q in $\mathbb{R}^1$ that realize M_ε in $O(nm + k^2n)$ time, where $m \leq n$.*

Proof. Correctness is given by Lemma 15. First note that the size of V derived from M_ε is $|V| = m$, and $|E| = O(m^2)$ due to the size of the adjacency matrix we use in Step (1), however, we need $O(k^2n)$ time to add all edges in G due to the size of the clique induced by the intervals with entry 1 in each row. The size of each clique is at most k^2. Thus $|E| = \min(m^2, k^2n)$. Steps (2–3) takes $O(|V| + |E|) = O\big(m + \min(m^2, k^2n)\big)$ for performing the BFS on G. Step (4) takes $O(|V|) = O(m)$ time using a counting sort per level of the

BFS. Step (5) takes $O(mn)$ time by processing each row of M_ε and partitioning the relevant intervals into their equivalence classes in $O(m)$ time. Step (6) takes $O(m)$ time. In step (7), the real line can be partitioned into $2m = O(m)$ cells. Verifying that all n points of P fall into the induced cell takes $O(nm)$ time. Thus, the algorithm's total runtime is $O(mn + k^2 n)$. ◀

References

1 Zachary Abel, Erik D Demaine, Martin L Demaine, Sarah Eisenstat, Jayson Lynch, and Tao B Schardl. Who needs crossings? Hardness of plane graph rigidity. In *32nd International Symposium on Computational Geometry (SoCG 2016)*, volume 51, pages 3:1–3:15, 2016.

2 Zachary Ryan Abel. *On folding and unfolding with linkages and origami.* PhD thesis, Massachusetts Institute of Technology, 2016.

3 Hugo A. Akitaya, Maike Buchin, Majid Mirzanezhad, Leonie Ryvkin, and Carola Wenk. Realizability of free space diagrams for 1D curves. In *38th European Workshop on Computational Geometry (EuroCG)*, 2022.

4 Hugo Alves Akitaya, Maike Buchin, Leonie Ryvkin, and Jérôme Urhausen. The k-Fréchet distance: How to walk your dog while teleporting. In *30th International Symposium on Algorithms and Computation, ISAAC 2019*, pages 50:1–50:15, 2019.

5 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5(1-2):75–91, 1995.

6 Helmut Alt, Christian Knauer, and Carola Wenk. Comparison of distance measures for planar curves. *Algorithmica. An International Journal in Computer Science*, 38(1):45–58, 2004.

7 Esther M. Arkin, Michael A. Bender, Erik D. Demaine, Martin L. Demaine, Joseph S.B. Mitchell, Saurabh Sethia, and Steven S. Skiena. When can you fold a map? *Computational Geometry*, 29(1):23–46, 2004. Special Issue on the 10th Fall Workshop on Computational Geometry, SUNY at Stony Brook.

8 Boris Aronov, Sariel Har-Peled, Christian Knauer, Yusu Wang, and Carola Wenk. Fréchet distance for curves, revisited. In *14th Annual European Symposium on Algorithms*, pages 52–63, 2006.

9 Peter F Ash and Ethan D Bolker. Recognizing Dirichlet tessellations. *Geometriae Dedicata*, 19:175–206, 1985.

10 Jérémy Barbay. Adaptive computation of the discrete fréchet distance. In *String Processing and Information Retrieval*, pages 50–60, 2018.

11 Hossein Boomari, Mojtaba Ostovari, and Alireza Zarei. Recognizing visibility graphs of polygons with holes and internal-external visibility graphs of polygons. *arXiv preprint*, 2018. `arXiv:1804.05105`.

12 Karl Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 661–670, 2014.

13 Karl Bringmann and Wolfgang Mulzer. Approximability of the discrete Fréchet distance. *Journal of Computational Geometry*, 7(2):46–76, 2016.

14 Kevin Buchin, Maike Buchin, Wouter Meulemans, and Wolfgang Mulzer. Four soviets walk the dog: Improved bounds for computing the fréchet distance. *Discrete & Computational Geometry*, 58(1):180–216, 2017.

15 Kevin Buchin, Jinhee Chun, Maarten Löffler, Aleksandar Markovic, Wouter Meulemans, Yoshio Okamoto, and Taichi Shiitada. Folding free-space diagrams: Computing the Fréchet distance between 1-dimensional curves (multimedia contribution). In *33rd International Symposium on Computational Geometry, (SoCG 2017)*, pages 64:1–64:5, 2017.

16 Kevin Buchin, Tim Ophelders, and Bettina Speckmann. SETH says: Weak Fréchet distance is faster, but only if it is continuous and in one dimension. In *Proc. 30th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2887–2901, 2019.

17 Maike Buchin, Anne Driemel, and Bettina Speckmann. Computing the Fréchet distance with shortcuts is NP-hard. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry (SOCG'14)*, pages 367–376, 2014.

18 Maike Buchin, Leonie Ryvkin, and Carola Wenk. On the realizability of free space diagrams. In *37th European Workshop on Computational Geometry (EuroCG)*, 2021.

19 Jean Cardinal and Udo Hoffmann. Recognition and complexity of point visibility graphs. *Discrete & Computational Geometry*, 57, January 2017.

20 Robert Connelly. Generic global rigidity. *Discrete & Computational Geometry*, 33(4):549, 2005.

21 Derek G Corneil, Hiryoung Kim, Sridhar Natarajan, Stephan Olariu, and Alan P Sprague. Simple linear time recognition of unit interval graphs. *Information Processing Letters*, 55(2):99–104, 1995.

22 Erik D. Demaine and Joseph O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, 2008.

23 Anne Driemel, Sariel Har-Peled, and Carola Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete & Computational Geometry*, 48:94–127, 2012.

24 Anne Driemel, Sariel Har-Peled, and Carola Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete & Computational Geometry. An International Journal of Mathematics and Computer Science*, 48(1):94–127, 2012.

25 Alon Efrat, Leonidas J. Guibas, Sariel Har-Peled, Joseph S.B. Mitchell, and T.M Murali. New similarity measures between polylines with applications to morphing and polygon sweeping. *Discrete & Computational Geometry*, 28(4):535–569, 2002.

26 Joachim Gudmundsson, Patrick Laube, and Thomas Wolle. Movement patterns in spatio-temporal data. In S. Shekhar and H. Xiong, editors, *Encyclopedia of GIS*. Springer-Verlag, 2007.

27 Timothy Franklin Havel. *The combinatorial distance geometry approach to the calculation of molecular conformation*. University of California, Berkeley, 1982.

28 Minghui Jiang, Ying Xu, and Binhai Zhu. Protein structure-structure alignment with discrete Fréchet distance. *Journal of Bioinformatics and Computational Biology*, 6(1):51–64, 2008.

29 Ross J Kang and Tobias Müller. Sphere and dot product representations of graphs. *Discrete Comput Geom*, 47:548–568, 2012.

30 Joseph Noggle. *The nuclear Overhauser effect*. Academic Press, 2012.

31 Günter Rote. Lexicographic Fréchet matchings. In *30th European Workshop on Computational Geometry*, 2014.

32 Leonie Ryvkin. *On distance measures for polygonal curves bridging between Hausdorff and Fréchet distance*. doctoralthesis, Ruhr-Universität Bochum, Universitätsbibliothek, 2021. doi:10.13154/294-8275.

33 James B Saxe. Embeddability of weighted graphs in k-space is strongly NP-hard. In *17th Allerton Conf. Commun. Control Comput., 1979*, pages 480–489, 1979.

34 Marcus Schaefer. Realizability of graphs and linkages. In *Thirty Essays on Geometric Graph Theory*, pages 461–482. springer, 2012.

35 R. Sriraghavendra, Karthik K., and Chiranjib Bhattacharyya. Fréchet distance based approach for searching online handwritten documents. In *Proc. 9th International Conference on Document Analysis and Recognition (ICDAR)*, pages 461–465, 2007.

A Omitted Proofs and Details from Section 3

▶ **Lemma 17.** *The problem of finding curves P and Q in $\mathbb{R}^2$ that realize an input diagram D_ε is in $\exists\mathbb{R}$.*

Proof. We reduce the problem to a system of real polynomial inequalities. The coordinate of vertices of P and Q are the variables. Each cell represents a constraint: The proof of Lemma 3.1 [3] gives a quadratic equation relating the endpoints of the corresponding segments.

Completely full cells require each segment to lie in the intersection of the ε-neighborhood of the other segment's endpoints. Completely empty cells require that each segment lies outside of the other segment's ε-neighborhood. All constraints can be expressed with a constant number of quadratic inequalities. ◀

Reduction description. Here, we detail the overview shown in Section 3. Refer to Figure 2. Note that every edge in Q corresponds to an edge of T. Every edge of T has two correspondents on Q. Every edge in P either corresponds to an edge of T or is part of an angle gadget. A free space cell is empty unless it corresponds to a pair of edges of P and Q that (i) correspond to the same edge, (ii) correspond to adjacent edges and their shared vertex is rigid, (iii) one is part of an angle gadget and the other corresponds to one of the edges in the angle represented by the gadget, or (iv) correspond to a pair of edges in T that were the result of a subdivision of an edge of G. In case (i), the corresponding cell has a $\pm 45°$ slab of width $\sqrt{2}$ depending on the direction of the traversal of the edge. In case (ii), if the rigid acute angle between the corresponding edges of G is $90°$, the corresponding cell has a quarter of a unit disk centered at the corner that corresponds to the rigid vertex. Else, the rigid angle is $180°$ the corresponding cell has a right isosceles triangle with two edges of length 1 and whose vertex incident to the right angle is at the corner of the cell that corresponds to the rigid vertex. In case (iii), as explained in the description of the angle gadget, the cells corresponding to long edges are empty. The cells between a short edge of P and an edge of Q incident to the angle will contain half of a unit disk so that the distance between the half-disk and the corner of the cell that correspond to the nonrigid vertex is 1. Finally, case (iv) is the same as case (ii) with a rigid angle of $180°$.

▶ **Lemma 18.** *Given a realization of P and Q, the subcurves that correspond to a rigid subgraph H exactly cover a rigid transformation of H.*

Proof. By construction, every pair of edges from P and Q that either correspond to the same edge of T, or to two adjacent edges of T that in turn correspond to edges in the same rigid subgraph H, define a partially full cell. Then, the claim follows by Lemma 1. ◀

B Omitted Details from Section 6

The full description of realizability of discrete curves in $\mathbb{R}^1$ is presented below:
DiscreteRealizabilityAlgo(M_ε) :

Step (1): Construct an abstract unit-interval graph G from M_ε whose rows are vertices in P and columns are vertices in Q: The vertex set of G represents the columns of M_ε, i.e., the interval of length 2ε centered at a point $q \in Q$. For every row ($p \in P$), the edge set of G is defined by including a clique between the columns (vertices in Q) that are filled with 1 in that row. We store the edges of G in an adjacency matrix. In the following we assume G connected, or else, we treat each component separately.

Step (2): Choose a left anchor as follows. As in [21], we get a set of candidates for left anchor by running a BFS search on G from an arbitrary vertex. The set contains the vertices with minimum degree in the deepest level of the BFS. The set of candidates must be in at most two equivalence classes, or else G is not an interval graph. We augment G by adding a new vertex v_0 connected to each vertex in one of the equivalence classes of candidates. We choose v_0 as a left anchor.

Step (3): Perform a BFS on G starting at v_0 to obtain a partial order of the vertices.

Step (4): Refine the partial order to get the global order by sorting the intervals at each level of the BFS under the criterion of $|\mathsf{Next}(N(v))| - |\mathsf{Prev}(N(v))|$. We denote the current partial order as D and use $<_D$, $>_D$ and $=_D$ to denote whether intervals appear in order, in reverse order, or are incomparable in D, respectively. By Theorem 2.2 in [21], a pair of vertices u and v with $u =_D v$ are indistinguishable. Thus a set of pairwise incomparable vertices in this partial order forms an equivalence class. Add a vertex v_f to the end of the order, connecting it to all vertices in the last equivalence class.

Step (5): Further refine the partial order as follows. We refer to such refinement as D'. For each row r in M_ε, let I_r be the set of intervals with entry 1 in r, and let $C' = C \cap I_r \neq \emptyset$ where C is an equivalence class. If there are $i \in I_r \setminus C$ and $c' \in C'$ where $i <_D c'$ (resp., $i >_D c'$), make $c' <_{D'} c$ (resp., $c' >_{D'} c$) for all $c \in C \setminus C'$.

Step (6): Extend the partial order defined by the BFS levels and D' to a global order breaking ties arbitrarily. Obtain the arrangement of unit intervals based on the global order and G. This can be done as in Theorem 3.2 in [21].

Step (7): Verify whether the produced arrangement is compatible with M_ε. Each row r specifies the existence of a cell where exactly the intervals with entry 1 intersect. If a cell specified by a row does not exist in the arrangement, return "NO". Otherwise, return "YES".