The k-Fréchet Distance: How to Walk Your Dog While Teleporting

Hugo Alves Akitaya
Department of Computer Science, Tufts University, Massachusetts, USA
hugo.alves_akitaya@tufts.edu

Maike Buchin
Department of Mathematics, Ruhr University Bochum, Germany
maike.buchin@rub.de

Leonie Ryvkin
Department of Mathematics, Ruhr University Bochum, Germany
leonie.ryvkin@rub.de

Jérôme Urhausen
Department of Information and Computing Sciences, Utrecht University, Netherlands
J.E.Urhausen@uu.nl

Abstract

We introduce a new distance measure for comparing polygonal chains: the k-Fréchet distance. As the name implies, it is closely related to the well-studied Fréchet distance but detects similarities between curves that resemble each other only piecewise. The parameter k denotes the number of subcurves into which we divide the input curves (thus we allow up to $k-1$ "teleports" on each input curve). The k-Fréchet distance provides a nice transition between (weak) Fréchet distance and Hausdorff distance. However, we show that deciding this distance measure turns out to be NP-hard, which is interesting since both (weak) Fréchet and Hausdorff distance are computable in polynomial time. Nevertheless, we give several possibilities to deal with the hardness of the k-Fréchet distance: besides a short exponential-time algorithm for the general case, we give a polynomial-time algorithm for $k = 2$, i.e., we ask that we subdivide our input curves into two subcurves each. We can also approximate the optimal k by factor 2. We then present a more intricate FPT algorithm using parameters k (the number of allowed subcurves) and z (the number of segments of one curve that intersect the ε-neighborhood of a point on the other curve).

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Design and analysis of algorithms; Theory of computation → Fixed parameter tractability

Keywords and phrases Measures, Fréchet distance, Hardness, FPT

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.50

Related Version A full version of the paper is available at `http://arxiv.org/abs/1903.02353`.

Funding *Hugo Alves Akitaya*: supported by NSF awards CCF-1422311 and CCF-1423615, and the Science Without Borders scholarship program.
Jérôme Urhausen: supported by the Netherlands Organisation for Scientific Research under project 612.001.651.

Acknowledgements We would like to thank Erik Demaine for contributing the key idea for proving hardness in the free space diagram in Section 3.1, as well as the organizers and other participants of the Intensive Research Program in Discrete, Combinatorial and Computational Geometry in Barcelona, 2018, for providing the perfect environment to meet other researchers.

1 Introduction

During the last decades, several methods for comparing geometrical shapes have been studied in a variety of applications, e.g., analyzing geographic data, such as trajectories, or comparing chemical structures like protein chains or DNA. The Fréchet distance has been well-studied in the past since it has proven to be helpful in several of the mentioned applications. The Hausdorff distance, another similarity measure, has also proven useful in applications and can be computed more efficiently. However, it provides us with less information by taking only the overall positioning of curves into consideration, not how they are traversed.

We introduce the k-Fréchet distance as a distance measure in between Hausdorff and (weak) Fréchet distance. This measure allows us to compare shapes consisting of several parts: we cover the input curves by at most k (possibly overlapping) subcurves each and ask for a matching of the subcurves such that each pair of matched subcurves has at most weak Fréchet distance ε (where $\varepsilon > 0$ is a given constant). Note that there are in fact two variants of the k-Fréchet distance: the *cover variant* described above and the *cut variant*, where we partition the input curves into k disjoint subcurves each. These subcurves are then matched if and only if their (weak) Fréchet distance is small. In this paper, by k-Fréchet distance we always refer to the cover variant and mention the cut variant only briefly.

Thus the new measure allows us to find similarities between curves that need to be cut and reordered to be similar under the Fréchet distance. For instance, this could be objects of rearranged pieces such as a set of trajectories of tourists visiting several sights in a city. If the k-Fréchet distance of two trajectories is small, the respective tourists used similar routes to get to the sights. For small k we can also conclude that the tourists visited many sights in the same order. Other examples would be chemical structures or handwritten characters and symbols. An example is displayed in Figure 1, where we compare three variants of writing the letter k by hand. Note that we deal with disconnected curves by concatenating the respective subcurves. Of course, we can easily identify that all three of them are k's by using the Hausdorff distance to compare them to a "generic" k, but the k-Fréchet distance provides us with more information: the 2-Fréchet distance between the second and the third k is large because the strokes are set differently. Those k's are unlikely to be written by the same person. The 3-Fréchet distance, however, is small, because the letter consists of at most 3 strokes in general.

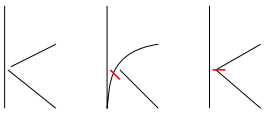

Figure 1 Three 'k's written in a different way. For the middle and the right one, the 2-Fréchet distance is large and the 3-Fréchet distance is small.

Characterizing the mentioned variants of the Fréchet distance next to the Hausdorff distance intuitively shows that the new distance measure bridges between weak Fréchet and Hausdorff distance. As is common for the Fréchet distance, we use the following analogy: we interpret our input curves as two paths, which have to be traversed by a man and a dog, each of them walking on one of the paths. For the (weak) Fréchet distance we ask for the length of the shortest leash so that man and dog can traverse their curves. They may choose their speeds independently. For the weak Fréchet distance, man and dog are allowed to backtrack.

The Hausdorff distance finds for each point on either curve the closest point on the other curve and takes the largest of the obtained distances. In terms of man and dog we do not ask for traversal as such, we simply need that for any fixed position on either path there is a

position on the other one such that man and dog can stand on their respective positions using a leash of fixed length. One could say they may "teleport" on their curves any number of times as long as both man and dog can reach all positions on their respective curves without exceeding the given maximum distance, i.e., the leash length. The k-Fréchet distance limits this number of teleports to a constant k (actually, we have $k-1$ teleports), so we want man and dog to traverse their paths piecewise. Note that we use the weak Fréchet distance as an underlying distance measure. As we picture man and dog to teleport, it is more natural to allow backtracking, especially since we allow any point on a curve to be the target point of a teleport. Imagine a subcurve oriented in the opposite direction than the subcurve closest to it: we would like to teleport the dog to the end of that curve and have it traverse the curve backward. Moreover, we do not ask to match the endpoints of our input curves.

Related work. Efficient algorithms were presented for computing the Fréchet distance and the weak Fréchet distance by Alt and Godau in 1995. They first introduced the concept of the free space diagram, which is key to computing this distance measure and its variants [3]. Following their work, numerous variants and extensions have been considered. Here we mention only a few results related to our work. Alt, Knauer and Wenk compared Hausdorff to Fréchet distance and discussed κ-bounded curves as a special input instance [4]. In particular, they showed that for convex closed curves Hausdorff distance equals Fréchet distance. For curves in one dimension Buchin et al. [6] proved the equality of Hausdorff and weak Fréchet distance using the well-known Mountain climbing theorem [18]. For computing the Hausdorff distance, Alt et. al. [2] gave a thorough overview. Buchin [9] gave the characterization of these measures in free space, which motivated our study of k-Fréchet distance.

For c-packed curves, Driemel, Har-Peled and Wenk presented a $(1+\varepsilon)$-approximation algorithm, which determines the Fréchet distance in near linear time [15]. For general polygonal curves, Buchin et al. [7] recently slightly improved the original algorithm of Alt and Godau, while Bringmann [5] showed that unless SETH fails no strongly subquadratic algorithm for the Fréchet distance exists. An interesting variant was presented by Gheibi et al.: they studied the weak Fréchet distance but minimized the length of the subcurves on which backtracking is necessary [17]. Buchin, Buchin and Wang studied partial curve matching, where they presented a polynomial-time algorithm to compute the "partial Fréchet similarity" [8], and a variation of this similarity was presented by Scheffer in [20]. Also, Driemel and Har-Peled defined a Fréchet distance with shortcuts [14], which was proven to be the first NP-hard variant of the Fréchet distance in [10].

Interestingly, both Hausdorff and (weak) Fréchet distance are computable in polynomial time. However, the k-Fréchet distance, as a distance measure that bridges between the two of them, proves to be NP-complete.

Overview. In the next chapter, we introduce and formally define the k-Fréchet distance. In Chapter 3, we determine its hardness in two steps: first, we prove NP-hardness of a simpler auxiliary problem to gain some intuition (Section 3.1) for the then following reduction. The most intricate part of our work is the construction of said reduction and analyzing its correctness, both of which are presented in Section 3.2. Finally, we present our algorithmic findings in Chapter 4. We give an XP-algorithm with parameter k, which even works in polynomial time for small k. A greedy approach leads to a 2-approximation on the optimal k. We then make use of two parameters, again the selection size k, and the parameter z, which indicates how "entangled" the input curves are, to construct our FPT algorithm.

2 Preliminaries

First we define the *Hausdorff distance* [4] for curves $P, Q\colon [0,1] \to \mathbb{R}^d$ as

$$\delta_{\mathrm{H}}(P,Q) = \max(\tilde{\delta}_H(P,Q), \tilde{\delta}_H(Q,P)), \text{ where}$$

$$\tilde{\delta}_{\mathrm{H}}(P,Q) = \max_{t_1\in[0,1]} \min_{t_2\in[0,1]} \|P(t_1) - Q(t_2)\|$$

denotes the directed Hausdorff distance from P to Q. By $\|\cdot\|$ we refer to the Euclidean norm in $\mathbb{R}^d$. Now recall the *Fréchet distance* [3]: For curves $P, Q\colon [0,1] \to \mathbb{R}^d$ it is given by

$$\delta_{\mathrm{F}}(P,Q) = \inf_{\sigma} \max_{t\in[0,1]} \|P(t) - Q(\sigma(t))\|,$$

where the reparametrizations $\sigma\colon [0,1] \to [0,1]$ range over all orientation-preserving homeomorphisms. A variant is the *weak Fréchet distance* δ_{wF} where both curves are reparameterised by σ and τ, respectively, which range over all continuous surjective functions.

As mentioned, the Fréchet distance is often illustrated by a man and a dog walking on two curves where both may choose their speed independently. For the Fréchet distance, man and dog may not backtrack, for the weak Fréchet distance they may. The (weak) Fréchet distance corresponds to the shortest leash length allowing them to traverse the curves.

A well-known characterization, which is key to efficient algorithms for computing both weak and (strong) Fréchet distance uses the free space diagram, which was introduced by Alt and Godau [3]. First we recall the free space F_ε:

$$F_\varepsilon(P,Q) = \{(t_1,t_2) \in [0,1]^2\colon \|P(t_1) - Q(t_2)\| \leq \varepsilon\}.$$

For piecewise-linear P and Q, the free space diagram puts this information into an $(n \times m)$-grid where n and m are the numbers of segments in P and Q respectively. For the rest of this paper we assume that $m = \mathcal{O}(n)$ to simplify runtime expressions.

The Fréchet distance of two curves is at most a given value ε if there exists a monotone path through the free space connecting the bottom left to the top right corner. For the weak Fréchet distance, this path need not be monotone. It may also start and end somewhere other than the corners of the diagram, as long as it touches all four boundaries.

We now define further terms regarding the free space diagram: A *component* of a free space diagram is a connected subset $c \subseteq F_\varepsilon(P,Q)$. A set S of components *covers* a set $I \subseteq [0,1]_P$ of the parameter space (corresponding to the curve P) if I is a subset of the projection of S onto said parameter space, i.e., $\forall x \in I\colon \exists c \in S, y \in [0,1]_Q\colon (x,y) \in c$. Covering on the second parameter space is defined analogously. This means the weak Fréchet distance is smaller than ε if there is one component in $F_\varepsilon(P,Q)$ that covers both parameter spaces. Similarly, the Hausdorff distance can be tested by checking whether the set of all components covers both parameter spaces. In this paper we extend this concept to also account for the number of components needed to cover the parameter spaces.

We define the *k-Fréchet distance* $\delta_{\mathrm{kF}}(P,Q)$ as the minimal ε such that there is a set of at most k components of $F_\varepsilon(P,Q)$ covering both parameter spaces. That is, we cover the curves P and Q by at most k pieces (i.e., subcurves) such that there is a matching of the subcurves where a matched pair has small weak Fréchet distance. Note that the subcurves may overlap. In the analogy, we allow man and dog to "teleport" on their respective curves, i.e., they may skip parts of their paths and come back later. We still ask for a complete traversal, but some parts of the curves may be traversed multiple times with teleports in between.

The decision problem for this distance measure asks whether for a fixed value of k, $\delta_{\mathrm{kF}}(P,Q)$ is smaller than or equal to a given ε. Naturally, for a fixed real $\varepsilon > 0$, we would

like to cut the curves into as few subcurves as possible (optimization version). By definition, the k-Fréchet distance lies in between the Hausdorff and the (weak) Fréchet distances:

$$\delta_{\mathrm{H}}(P,Q) \leq \delta_{\mathrm{kF}}(P,Q) \leq \delta_{\mathrm{wF}}(P,Q) \leq \delta_{\mathrm{F}}(P,Q).$$

Also, the k-Fréchet distance decreases as k increases: for $k = 1$ it equals the weak Fréchet distance, whereas for k sufficiently large, e.g., $k \geq n^2$, it equals the Hausdorff distance.

Figure 2 illustrates this property. The diagram on the left corresponds to a fixed ε_1. We observe that there is one connected component in the free space $F_{\varepsilon_1}(P,Q)$ that projects surjectively onto both parameter spaces. We therefore have $\varepsilon_1 \geq \delta_{\mathrm{wF}}$ $(= \delta_{1\mathrm{F}})$. The diagram in the middle depicts $F_{\varepsilon_2}(P,Q)$ for a value ε_2 slightly smaller than ε_1. In that case two components cover the parameter spaces, which means $\varepsilon_2 \geq \delta_{2\mathrm{F}}$. The free space $F_{\varepsilon_3}(P,Q)$ shown on the right for an ε_3 smaller than ε_2 consists of three components and all three are necessary to cover the parameter spaces. Furthermore, reducing the value of ε_3 even more would not split up the components into smaller subcomponents, but would just result in the set of all components not covering the parameter spaces any more. So we have $\varepsilon_3 \geq \delta_{\mathrm{H}} = \delta_{3\mathrm{F}}$.

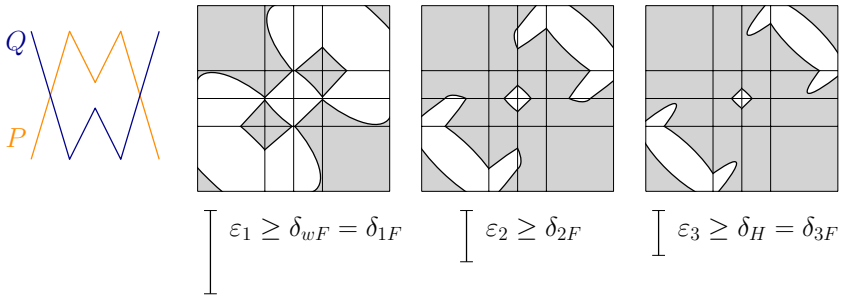

Figure 2 Comparison of weak Fréchet, 2-Fréchet and Hausdorff distance of curves P and Q.

3 Hardness results

In this section, we prove that deciding the k-Fréchet distance for fixed ε is NP-hard.

To give some intuition for the later proof, we first present a reduction from the well-known 3-SAT problem to the problem of covering two sides of a rectangle by selecting a number of smaller rectangles, or boxes, that are situated inside. This problem (we call it the box problem) mimics selecting the components in the free space to cover the parameter spaces. However, we do not ask to find curves that realize this specific free space.

Afterwards we reduce from rectilinear monotone planar 3-SAT [13] to prove hardness of the actual k-Fréchet distance problem.

3.1 Gaining intuition: The box problem

We want to reduce from the following classical NP-hard satisfiability problem [16]:

3-SAT:
Input: a boolean formula with n variables written as a conjunction of m clauses, where a clause is a disjunction of at most 3 literals;
Output: "Yes" if there exists a satisfying variable assignment, "No" otherwise.

Box problem:
INPUT: a set A of aligned, interior-disjoint rectangles b_i, their bounding box B, $k \in \mathbb{N}$;
OUTPUT: "Yes" if there exists a selection of at most k rectangles from A such that their union surjectively projects onto the bottom and left boundary of B, "No" otherwise.

Given any instance of a 3-SAT formula, we build a bounding box B containing a number of boxes b_i such that we can find a covering selection of size k if and only if there is an assignment for the formula that outputs true. A *covering selection* of boxes is a subset of the b_i that projects surjectively onto the bottom and left boundaries of B. For this we build boxes b_i that correspond to the variables and any satisfying assignment of the variables can be directly "translated" into a covering selection of the b_i.

First, note that we assume that no clause contains duplicates, i.e., no clause is of the form $v \vee v \vee w$. The duplicates can be deleted without changing the boolean function induced by the formula. Note that clauses of the form $v \vee \neg v \vee w$ are allowed. Additionally, we require that throughout the formula each literal appears at least once, i.e., each variable appears at least once in its positive and in its negated form. For each variable v where this is not the case we add the clause $v \vee \neg v$ (colored dark green in Figure 3). These clauses are always fulfilled and therefore do not change the output of our boolean formula. We add at most n clauses in this way, which means that the size of the formula only changes polynomially in the input size.

Now we give the detailed construction of our box problem instance derived from some 3-SAT formula: Let $V = \{v_1, \dots, v_n\}$ be the set of variables and let $C = \{c_1, \dots, c_m\}$ be the set of clauses. For each variable v_i, let a_i^+ (respectively a_i^-) be the number of clauses in which v_i appears positive (respectively negatively), and let $\{c_{i,1}^+, c_{i,2}^+, \dots, c_{i,a_i^+}^+\}$ (respectively $\{c_{i,1}^-, \dots, c_{i,a_i^-}^-\}$) be the set of clauses in which v_i appears positive (respectively negatively). Additionally we define the sums $s_i^+ = \sum_{k=1}^{i} a_k^+$ and $s_i^- = \sum_{k=1}^{i} a_k^-$.

In the following we describe the placement of boxes, which is depicted in Figure 3. The number of rows and columns needed for the different gadgets is indicated in the figure. A box (x, y, w, ℓ) designates the axis-aligned rectangle with unit height and width w whose bottom left corner has coordinates $(x, y) \in \mathbb{R}^2$ with label ℓ. The labels are later used in the proof of correctness.

Variable gadget. For each variable v_i, we place two boxes $(i, i, 1, \neg v_i)$ and $(i, i+n+s_n^+, 1, v_i)$, and no other boxes are placed over the interval $(i, i+1)$ of the bottom boundary. That way, in order to cover said interval, at least one of those two boxes has to be chosen.

Split gadget. The split gadget ensures that we can propagate the assignment of a variable onto all clauses the variable takes part in. We build the splits used for the positive occurrences of the variables first. For each variable v_i, we place the box $(1 + n + s_{i-1}^+, i, a_i^+, v_i)$ and the boxes $(n + s_{i-1}^+ + j, n + s_{i-1}^+ + j, 1, \neg v_i)$, for $j \in \{1, \dots, a_i^+\}$. For negated occurrences of $v_i \in V$ we place the box $(1 + n + s_n^+ + s_{i-1}^-, n + s_n^+ + i, a_i^-, \neg v_i)$ and the boxes $(n + s_n^+ + s_{i-1}^- + j, 2n + s_n^+ + s_{i-1}^- + j, 1, v_i)$, for $j \in \{1, \dots, a_i^-\}$.

Clause gadget. We assign to each clause c_i the unit interval on the bottom boundary of B starting at $I(c_i) = n + s_n^+ + s_n^- + i$. For each literal of a clause c_i we place a box labeled with the respective literal above the unit interval $[I(c_i), I(c_i) + 1]$. To be precise, for each $v_i \in V$ we place the boxes $(I(c_h), n + s_{i-1}^+ + j, 1, v_i)$, for $j \in \{1, \dots, a_i^+\}$ and $h \in \{1, \dots m\}$ where $c_h = c_{i,j}^+$, and $(I(c_h), 2n + s_n^+ + s_{i-1}^- + j, 1, \neg v_i)$, for $j \in \{1, \dots, a_i^-\}$, $h \in \{1, \dots m\}$ where this time $c_h = c_{i,j}^-$.

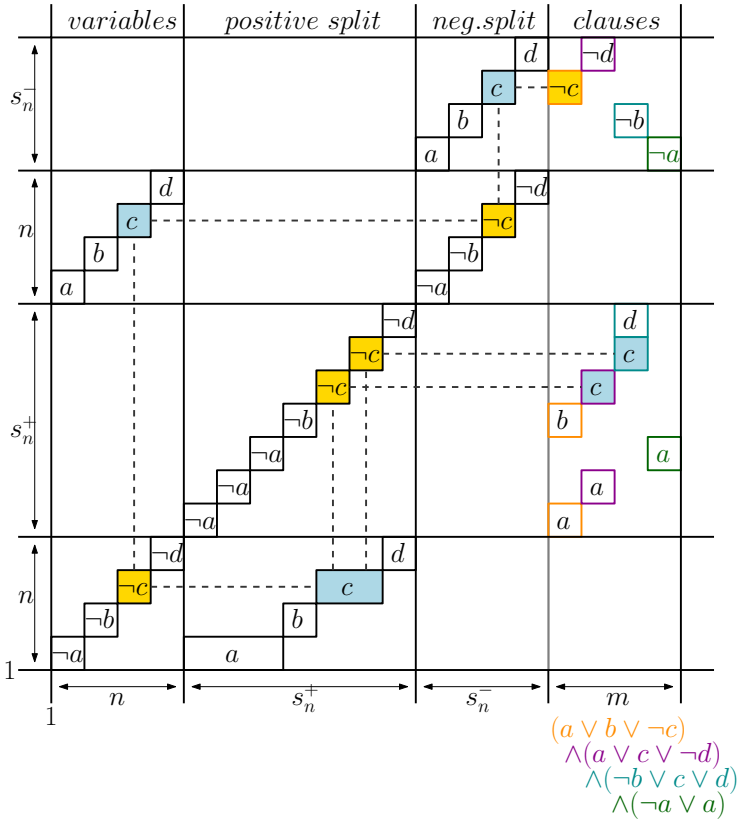

Figure 3 Construction of the box problem instance and propagation of assignment.

Overall, we have $4n + 2(m_1 + 2m_2 + 3m_3)$ boxes, where m_i is the number of clauses with i variables (and therefore $m_1 + m_2 + m_3 = m$). Each unit interval $(i, i+1)$ with $i \in \{1, \ldots, 2n + s_n^+ + s_n^-\}$ on the left boundary of B can be covered by exactly two different boxes. The same holds for every unit interval $(i, i+1)$ with $i \in \{1, \ldots, n + s_n^+ + s_n^-\}$ on the bottom boundary. Note that for all these unit intervals, one of the boxes is labeled with a variable and the other one is labeled with the negated version of that variable, i.e., one box is labeled v and the other one $\neg v$. Each Interval $I(c)$ on the bottom boundary can be covered by as many boxes as the clause c contains literals. The labels of these boxes correspond to the variables contained within this clause. We set the bounding box B as the axis-aligned rectangle spanned by the points $(1, 1)$ and $(1 + n + s_n^+ + s_n^- + m, 1 + 2n + s_n^+ + s_n^-)$ and we set $k = 2n + m_1 + 2m_2 + 3m_3$ so only half the boxes can be chosen. For a given boolean formula, the set of boxes defined above can be determined in polynomial time.

▶ **Theorem 1.** *The box problem is NP-hard.*

Proof. First we prove that the box problem as constructed above has a solution if and only if the input 3-SAT formula has a variable assignment such that it evaluates to `true`.

"⇐". Let $f : V \to \{\texttt{true}, \texttt{false}\}$ be an assignment of the variables that satisfies the 3-SAT formula. We set $S = \{\text{boxes } (x, y, w, v) \mid f(v) = \texttt{true}\} \cup \{\text{boxes } (x, y, w, \neg v) \mid f(v) = \texttt{false}\}$. The set S projects surjectively onto the bottom and left boundary of the bounding box B because each unit interval on the left boundary is covered by exactly one box. For most of the bottom boundary we also have that each interval is uniquely covered, but for the clauses columns we allow that more than one box per unit interval is chosen (i.e., more than one corresponding literal is set to `true`).

"⇒". Let S be a minimal set of boxes that covers the boundaries of the bounding box B with $|S| = k$. This means that each unit interval on the left boundary of B is covered by exactly one box. Due to the position of the boxes, this means that for each variable v either all boxes labeled v or all boxes labeled $\neg v$ have been chosen. This induces an assignment of the variable v, i.e., v is set to `true` if the boxes labeled v have been chosen and else v is set to `false`. Note that the selection S covers the box B. Therefore, for each clause c one of the boxes that can cover $I(c)$ is an element of the selection S. It follows that the assignment of variables induced by S fulfills the formula.

Above we showed the NP-hardness of the box problem. The box problem is in fact even NP-complete since for a given subset S of boxes one can test if the bounding box B is covered by simply marking the covered intervals, which can be done in polynomial time. ◀

We can interpret the box problem as the problem of finding a selection of components in the free space that cover the parameter spaces. The small boxes can be seen as bounding boxes of actual components (for the projection there is no difference) and the bottom and left boundary of the large box B correspond to the respective parameter spaces. The above hardness proof, especially the construction of the boxes, provides us with the key ideas to prove hardness of the k-Fréchet distance. Next, we construct actual curves where certain intervals on the parameter spaces of the free space diagram each have two components that could cover them. As with the box problem, the choice we make for one of those intervals determines the choices for other intervals as we still need to ensure that the selection size is minimal in the end. The propagation of choices works in the same manner for the box problem as for the k-Fréchet distance problem.

3.2 Reduction for the k-Fréchet distance

We use the following variant of the 3-SAT problem in this subsection.

Rectilinear monotone planar 3-SAT:
Input: a 3-SAT formula with only all positive or all negated variables per clause, embedded as a graph with rectilinear, non-crossing edges; variables are drawn as vertices on a horizontal line, positive clauses are vertices drawn above this line and negative clauses are drawn below;
Output: "Yes" if there exists a satisfying assignment for the variables, "No" otherwise.

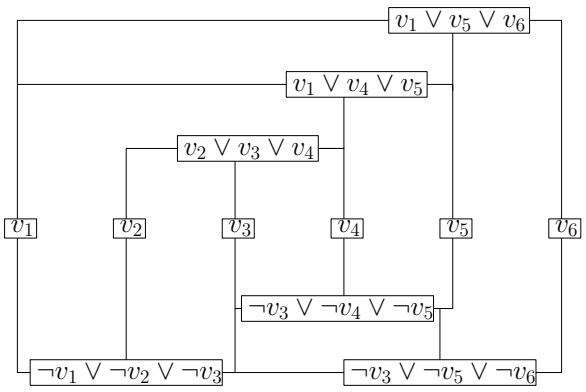

Figure 4 Instance of rectilinear monotone planar 3-SAT.

Note that we assume that each variable appears in at least one positive and one negative clause. Otherwise, we could simply define the occurring literal to be `true` (or `false`, respectively) and omit the clauses the literal appears in.

We can draw any graph corresponding to such a 3-SAT formula on a grid, see, e.g., Figure 4, which is useful when constructing and analyzing our curves. Since rectilinear monotone planar 3-SAT is NP-hard [13], we prove hardness of the k-Fréchet distance problem by reducing from it.

***k*-Fréchet distance problem:**
INPUT: Two polygonal curves P and Q, a distance ε and a natural number k;
OUTPUT: "Yes" if there exists a selection of at most k components in the free space diagram F_ε such that their union projects surjectively onto both parameter spaces, "No" otherwise.

Our goal is to construct two curves P (yellow) and Q (blue) that mimic any input instance of a rectilinear monotone planar 3-SAT graph and show that in the free space resulting from these curves we can find a covering selection of size k if and only if there exists a satisfying assignment for the formula. The detailed construction can be found in the full version of this paper [1].

Overall we create wire and clause gadgets to represent variables and clauses, where wires correspond to the edges of the input graph. Wire gadgets allow a boolean choice that is propagated consistently throughout the wire. Clause gadgets test whether at least one incoming wire carries an appropriate choice.

Figure 5 shows a wire gadget and how it is used. Both curves consist of two long parallels, which we call *base parts* of the curves, and *spikes*, which are the horizontal segments in the figure. The spikes are formed by taking a 90 degree turn from the base part and traversing the spike segment back and forth. The base parts are not particularly relevant for the analysis because the segments forming them can only be covered by larger components that are always part of any covering selection. The value ε is chosen such that two adjacent spikes are just within distance ε. It follows that the spikes induce components that are similar to the boxes of Subsection 3.1. We say that a spike s is *covered* by an adjacent spike t of the other curve if the component of the free space diagram that covers the two intervals induced by these spikes is chosen for the covering selection. In the end, we choose k such that each blue spike in any gadget can only be covered by one single adjacent yellow spike. The choice for blue spikes must be consistent along the wire to preserve minimality of k, and it encodes the assignment of the corresponding variable.

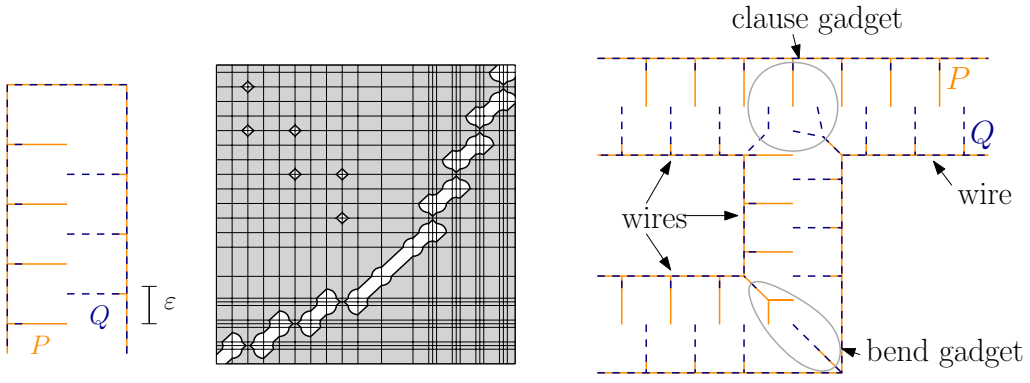

Figure 5 (Left and middle) The wire gadget and its corresponding free space diagram. Note that we connected the curves to give a small example, but the horizontal segment on top is not part of the wire itself. (Right) A part of the construction where wires connect other gadgets.

As displayed in Figure 5, the clause gadget features one yellow spike that can be covered by either one of the three blue spikes within its ε-neighborhood. Which one of their neighboring

yellow spikes the blue ones cover is determined by the variable assignment and propagated throughout the wire, so if at least one of the variables is set to `true`, the yellow spike at the center of the clause is covered.

Next, we need a number of other gadgets, too. As mentioned, the wires correspond to edges in the rectilinear monotone planar 3-SAT instance. To draw them coherently we need to make sure we can make 90 degree turns (so-called *bends*, see the right-hand side of Figure 5) and do T-crossings, i.e., *split* a wire into two.

We need to treat remaining difficulties: first of all, there is a *connection gadget* that enables us to connect the opposite base parts of P and Q, respectively. The resulting curves are closed, which we solve by applying the *scissor gadget*. Finally, we may need to change which of the curves has spikes on a specific side to draw the other gadgets consistently, so we also built a *color gadget* to "switch" the color pattern of the spikes.

At last, we want to connect all gadgets such that the resulting curves follow the embedding of the input graph G. Recall that the input is a grid embedding. We first scale the grid by a factor of 2^{10} to place all gadgets consistently. Note, that we have to deal with 2-clauses and take into account that our split gadget is directed, so we need to have some space for workarounds. Afterwards we can draw the curves' vertices on grid points only. Consider the input graph G. We want to traverse all edges of G twice, once per inner, once per outer base parts. To do so, we have to "walk around" each face of G. To switch between faces we use connection gadgets. We obtain a traversal order of the faces by computing a minimum spanning tree of the dual graph, see [1] for a detailed description.

Finally, it remains to prove that our construction works in the sense that the curves have k-Fréchet distance ε if and only if the specific 3-SAT instance is satisfiable.

First, we note that the complexity of our curves is polynomial in the size of our input instance: the numbers of variables and clauses, but also the number of splits and the length of the edges determine the number of spikes and therefore also the number of components in the free space diagram. A spike induces either two or three components, depending on whether it is part of a specific gadget, i.e., a clause, or not. In addition, the gadgets induce a number of components, called *clutter*, that are always part of a covering selection. Some gadgets also induce a constant number of unnecessary components that are never chosen.

Our goal is to cover the parameter spaces with k components. We definitely need to select all clutter components and we need to cover all spikes, therefore we need to select (at least) one component per spike. We set k to be the number of clutter components plus the number of blue spikes (spikes of Q). It follows that each blue spike can only be covered once, which ensures that choices are propagated.

▶ **Theorem 2.** *It is NP-hard to decide whether $\delta_{\mathrm{kF}}(P,Q) \leq \varepsilon$ for given polygonal curves P and Q, integer k, and $\varepsilon > 0$ where δ_{kF} denotes the k-Fréchet distance.*

For the full proof, we refer to the arXiv-Version of this paper [1]. For the reduction constructed above, the following holds: given a satisfying assignment for the input formula, we know which components to select: apart from all clutter components, we have to decide how to cover the blue spikes. This choice is implied by the assignment and propagated throughout the gadgets.

Given a selection of components, we need to backtrack our choices throughout the wires and other gadgets to determine how the blue spikes are covered. Depending on this choice, we know whether the corresponding variable has to be set to `true` or to `false`. Thus we derive our assignment for the 3-SAT formula and complete the proof of NP-hardness.

We can test in polynomial time whether the union of a selection of components covers the parameter spaces. Thus the problem of deciding the k-Fréchet distance lies in NP.

4 Algorithms

In this chapter, we begin by presenting a preprocessing algorithm, which applies to the following algorithmic approaches: First, we can find a covering selection of at most size k in exponential time and describe how to approximate k by factor 2 (Section 4.2). Then, we describe an FPT-algorithm for finding an optimal covering selection in Section 4.3. Note that the XP-algorithm as well as the FPT-algorithm are designed to solve the decision problem, but we can also optimize k by repeating the decision problem solving algorithm for different values of k by performing a parametric search on the reasonable values for k, similar to the algorithm for the Fréchet distance for polygonal curves by Alt and Godau [3].

4.1 Preprocessing

First, we observe two preprocessing strategies, which can be applied before entering any of our algorithms. In any case we start by computing the free space diagram, which takes quadratic time. In the free space diagram it is easy to identify all *necessary* components: any component that covers an interval of one of the parameter spaces uniquely (i.e., there is no other component covering the exact same interval) is necessarily chosen for an output selection. Such components can be found in $\mathcal{O}(n \log n)$ time using a scan. Furthermore, it is possible to rule out all *redundant* components. A component is called redundant if and only if it is completely contained in the bounding box of a different component (but there could be more than one such component with a sufficiently large bounding box). This case can also be detected via scans. Thus our preprocessing needs quadratic time. However, it does not improve the size of the input (being the complexity of the free space) nor the resulting runtime of any of the presented algorithms asymptotically.

4.2 XP-algorithm and approximation

We start by giving the more straight-forward approaches. First, we present an XP-algorithm.

▶ Remark 3. The k-Fréchet distance can be decided in $\mathcal{O}(k \cdot n^{2k})$ time for constant k.

The brute force approach simply checks for all selections of k components of the free space whether their joint projections cover both parameter spaces surjectively. That means we have to check at most $\binom{n^2}{k}$ possible combinations of components resulting in a runtime of $\mathcal{O}(k \cdot n^{2k})$ for fixed k, which is of course only feasible for very small k. Therefore we can compute the answer to the decision problem for the cover distance with $k = 2$ in $\mathcal{O}(n^4)$. Since $\binom{m}{k} \leq 2^m$ holds for any $m > k$, our runtime is upper-bounded by $\mathcal{O}(n \cdot 2^{n^2})$ for general k. We can also approximate the size of an optimal solution.

The main idea of our algorithm is to find minimal covering selections for each parameter space individually and combine those selections into an overall solution in the end. We can find both selections covering only a single parameter space by applying a greedy technique.

Given the free space diagram, we first project all components onto the parameter spaces. We get two sets of intervals, one covering the first parameter space (we store these intervals in the list L_P, see Figure 6) and one for the second parameter space (stored in L_Q). So one component projects onto two intervals, one on each parameter space (and thus one per list). We store the information on which two intervals stem from the same component accordingly.

Now we simply want to select a minimum number of intervals whose union equals the unit interval, i.e., the parameter space. We deal with each parameter space on its own as follows: we sort the lists L_P (and L_Q) by left endpoint. Now, per list, starting at 0, we make a greedy choice and select the interval (among the intervals starting at 0) with the

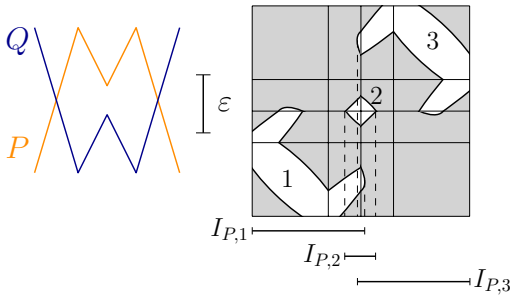

Figure 6 The projection onto the first parameter space and the resulting elements of L_P.

rightmost endpoint, say r_1. Here we recurse, i.e., we take r_1 as new start point and again search among the intervals covering r_1 (i.e., intervals starting at or to the left of r_1) for the one with the rightmost endpoint. As soon as we select an interval with 1 as endpoint we have found a minimal covering selection. To see that our greedy strategy is optimal, observe that the algorithm proceeds from left to right maintaining the following invariant: at any time we selected a minimum number of intervals to cover the parameter space from its left boundary to the current position.

As output we have two selections of intervals, S_P and S_Q. The intervals correspond to components. We build the union of both lists, taking into account that an interval in S_P may belong to the same component as an interval of S_Q, and output the selection of components S that contributed at least one of the chosen intervals.

The worst case that might occur is the following: all of the intervals we selected during the greedy procedures correspond to different components in the free space, so that the union of our selections is of size $|S_P| + |S_Q|$. A different selection of size $|\hat{S}| = \max(|S_P|, |S_Q|)$ might cover both parameter spaces but is not detected by the greedy scan. Schäfer proves that the approximation factor 2 is indeed tight [19].

Finally, we consider the runtime: computing the free space takes quadratic time. Sorting the lists adds another logarithmic factor while the greedy selection routine takes linear time in the number of intervals. Hence we get an overall runtime of $\mathcal{O}(n^2 \log n)$.

▶ **Theorem 4.** *The algorithm described above runs in $\mathcal{O}(n^2 \log n)$ time and finds a selection of components that covers both parameter spaces if and only if one exists. A found selection contains at most twice the minimum number of components needed.*

4.3 Fixed-parameter tractability

Lastly, we present an algorithm for deciding whether $\delta_{\mathrm{kF}}(P, Q) \leq \varepsilon$ for given ε and k. The runtime of our algorithm is polynomial in the complexity of our curves P and Q, but exponential in the two parameters k (the selection size) and z (the neighborhood complexity).

We define the *neighborhood complexity* as the maximum number of segments of one curve that intersect with the ε-neighborhood of any point of the other curve. In the free space diagram we get that each horizontal or vertical line intersects at most z components. Note that this requirement is similar but not directly related to c-packedness. The first counts the number of segments of one curve within a fixed distance of any point of the other curve. In contrast, c-packedness bounds the length of one curve within a ball of arbitrary radius.

The idea is the following: we build two directed bounded search trees (as described in Chapter 3 of [12]) to create selections of components of size at most k. Each search tree represents the projection of the free space onto one parameter space, see Figure 7 below. A

node corresponds to a component in the free space (or rather the interval on the respective parameter space it covers) and a path encodes a selection that covers the interval $[l, r]$. By l we denote the left boundary point of the interval corresponding to the root of the path and r is the right boundary point of the interval of the bottommost node (e.g. a leaf). We call a selection or a path *feasible* if the union of the (at most k) components it encodes covers the respective parameter space. From the first tree, T_P, we are able to extract all feasible selections which cover the parameter space corresponding to curve P, feasible selections of the second tree, T_Q, cover the other parameter space. In the end, we compare and/or combine a feasible selection of T_P with a feasible selection of T_Q to get a selection S that contains no more than k components, so that its union covers both parameter spaces.

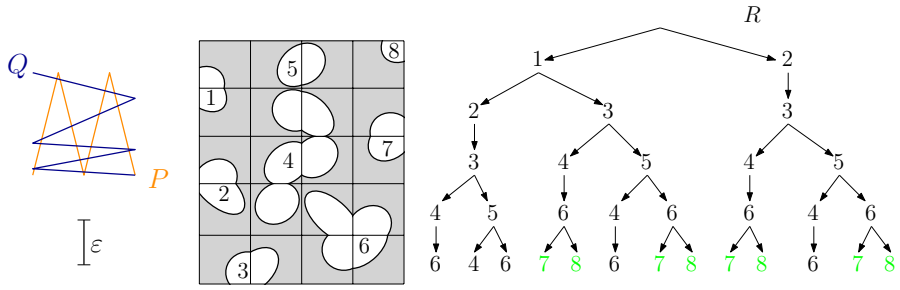

Figure 7 Curves P, Q, their free space diagram and the resulting bounded search tree T_P. Leaves of feasible paths are marked in green.

More formally, we build two trees of depth k and branching factor z. Consider the tree T_P. The root is labeled by the left boundary point of the parameter space of P (we assume w.l.o.g. that the bottom boundary of the free space diagram corresponds to P). Now we use a sweep line initialized at the left boundary of the free space diagram. We assign a node in the tree to all components intersecting the sweep line, i.e., the root has as many children as there are components touching the left boundary of the free space diagram. The sweep line moves to the right. Whenever the sweep line is tangent to a component, one of two cases occur: if it touches the leftmost point of a component it becomes *active*, i.e., the sweep line continues to intersect this component when moving further to the right; if the line touches the rightmost point of the component, it becomes *inactive* (so the sweep line just stops to intersect it). In the first case, nothing immediate happens to the tree, in the latter case, if the tangent component already has a node in the tree, we insert new nodes: each node corresponding to the tangent component gets assigned as many children as there are other currently active components. By definition, a node can never have more than z children. Note that some (small) components may not get assigned any nodes in one tree. Also, with every node we store its depth (the root has depth 0) and stop assigning children at depth k - or as soon as a component touches the right boundary of the free space diagram. If a leaf v_l corresponds to a component touching the right boundary, the path from the root to v_l encodes a feasible selection of components for T_P. Other selections are called *non-feasible*. The second tree T_Q is built analogously by sweeping from bottom to top.

We store the feasible selections obtained from T_P and T_Q in sorted lists L_P^T and L_Q^T. For each pair of selections $S_{P,i}$, $S_{Q,j}$, where $1 \leq i, j \leq z^k$, we test whether $|S_{P,i} \cup S_{Q,j}| \leq k$ and output this union if the answer is positive.

▶ **Theorem 5.** *The algorithm described above returns a selection S of k components in the free space that covers both parameter spaces if and only if such a selection exists. Hence it solves the decision problem for the k-Fréchet distance in time $\mathcal{O}(nz + kz^{2k})$.*

Proof. Our algorithm treats all possible selections of size at most k per parameter space and combines all these, hence it necessarily finds a valid solution if and only if one exists.

For the first step, we compute the free space, which takes $\mathcal{O}(n \cdot z)$ time. Building the trees takes $\mathcal{O}(z^k)$ time since we are limited to depth k and insert at most z children per node. Note that any operation in the free space diagram, such as detecting components that cover a boundary or projecting them onto the boundaries to find the intervals they cover, can be done in $\mathcal{O}(nz)$ time. We obtain at most z^k selections per search tree. Each selection is stored as the sorted set $S_{P,i}$, respectively $S_{Q,i}$, by encoding each component as an integer. We then sort both lists of selections lexicographically. During the sorting we might detect duplicates, which we discard immediately. We then compare each selection of the first list L_P^T to each selection of L_Q^T, taking time k per comparison. For any selection smaller than k we can test whether its union with a selection of the other list is still a solution, i.e., whether the unified selection does not have more than k integers. All in all, we have a runtime of $\mathcal{O}(zn + z^k + z^k \cdot k \log k + k(z^k)^2) = \mathcal{O}(zn + kz^{2k})$. ◀

By combining a greedy approach with building an interval tree, Schäfer improved the runtime of our FPT algorithm to $\mathcal{O}(nz + k \cdot (\log n + z) \cdot z^k)$ in [19].

5 Conclusion

We present a novel variant of the Fréchet distance for polygonal chains that allows to compare objects of rearranged pieces. We ask for k (possibly overlapping) subcurves per input curve that have pairwise small weak Fréchet distance. Thus, the k-Fréchet distance provides a transition between weak Fréchet ($k = 1$) and Hausdorff distance (k sufficiently large).

But as we prove, deciding the k-Fréchet distance of two polygonal curves is NP-hard. However, we were able to tackle the computational challenge from different angles: we give an XP-algorithm depending on k, approximate k by factor 2 and present an FPT-algorithm.

As mentioned in the introduction, there is a second variant of defining the k-Fréchet distance we call the "cut version": instead of allowing overlapping subcurves, we cut the curves into pieces and search for a matching between these pieces. The NP-hardness proof presented at EuroCG2018 [11] holds for this variant. However, the algorithmic approaches only work for the variant we discuss in this paper (we call it the "cover variant"). Finding algorithmic approaches for the cut version of the k-Fréchet distance is work in progress.

References

1 Hugo A. Akitaya, Maike Buchin, Leonie Ryvkin, and Jérôme Urhausen. The k-Fréchet distance. *CoRR*, 2019. `arXiv:1903.02353`.

2 Helmut Alt, Peter Braß, Michael Godau, Christian Knauer, and Carola Wenk. Computing the Hausdorff distance of geometric patterns and shapes. In *Discrete and computational geometry*, volume 25 of *Algorithms Combin.*, pages 65–76. Springer, 2003.

3 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *Internat. J. Comput. Geom. Appl.*, 5:75–91, 1995.

4 Helmut Alt, Christian Knauer, and Carola Wenk. Comparison of distance measures for planar curves. *Algorithmica*, 38(1):45–58, 2004.

5 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.

6 Kevin Buchin, Maike Buchin, Christian Knauer, Günther Rote, and Carola Wenk. How Difficult is it to Walk the Dog? In *23rd European Workshop on Computational Geometry (EuroCG)*, pages 170–173, 2007.

7 Kevin Buchin, Maike Buchin, Wouter Meulemans, and Wolfgang Mulzer. Four Soviets Walk the Dog: Improved Bounds for Computing the Fréchet Distance. *Discrete and Computational Geometry*, 58:180–216, 2017.

8 Kevin Buchin, Maike Buchin, and Yusu Wang. Exact algorithms for partial curve matching via the Fréchet distance. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, pages 645–654, 2009.

9 Maike Buchin. *On the Computability of the Fréchet Distance Between Triangulated Surfaces*. PhD thesis, Free University Berlin, Institute of Computer Science, 2007. URL: `http://www.diss.fu-berlin.de/diss/receive/FUDISS_thesis_000000002618`.

10 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. ACM, 2014. `doi:10.1145/2582112.2582144`.

11 Maike Buchin and Leonie Ryvkin. The k-Fréchet distance of polygonal curves. In *34th European Workshop on Computational Geometry (EuroCG)*, 2018. URL: `conference.imp.fu-berlin.de/eurocg18/`.

12 Marek Cygan, Fedor V. Fomin, Ł ukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, 2015.

13 Mark de Berg and Amirali Khosravi. Optimal binary space partitions for segments in the plane. *Internat. J. Comput. Geom. Appl.*, 22:187–205, 2012.

14 Anne Driemel and Sariel Har-Peled. Jaywalking your Dog - Computing the Fréchet Distance with Shortcuts. *CoRR*, 2011. `arXiv:1107.1720`.

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

16 Michael R. Garey and David S. Johnson. *Computers and intractability*. W. H. Freeman and Co., San Francisco, Calif., 1979.

17 Amin Gheibi, Anil Maheshwari, Jörg-Rüdiger Sack, and Christian Scheffer. Minimum backward Fréchet distance. In *Proceedings of the 22Nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '14, pages 381–388. ACM, 2014. `doi:10.1145/2666310.2666418`.

18 Jacob E. Goodman, János Pach, and Chee-K. Yap. Mountain climbing, ladder moving, and the ring-width of a polygon. *Amer. Math. Monthly*, 96:494–510, 1989.

19 Peter Schäfer. Untersuchungen zu Varianten des Fréchet-Abstands. Master's thesis, Fernuniversität Hagen, 2019.

20 Christian Scheffer. More Flexible Curve Matching via the Partial Fréchet Similarity. *Int. J. Comput. Geometry Appl.*, 26:33–52, 2016.