On Computing the (Exact) Fréchet Distance with a Frog

Jacobus Conradi
Department of Computer Science, University of Copenhagen, Denmark

Ivor van der Hoog
IT University of Copenhagen, Denmark

Eva Rotenberg
IT University of Copenhagen, Denmark

Abstract

The continuous Fréchet distance $\mathcal{D}_F(\pi, \sigma)$ between two polygonal curves π and σ is classically computed by exploring the *free space diagram* over the two curves. [SoCG'25] recently proposed a radically different approach: they approximate $\mathcal{D}_F(\pi, \sigma)$ by computing paths in a discrete graph that models a joint traversal of π and σ, recursively bisecting edges until the discrete distance converges to the continuous one. They implement their "frog-based" technique, and claim that it yields substantial practical speedups compared to the state-of-the-art implementations.

In this paper, we revisit this technique. We observe that, in its current form, it has three limitations: (i) it does not use exact arithmetic, (ii) its recursive bisection introduces the required monotonicity events to realise the Fréchet distance only in the limit, and (iii) it applies a heuristic simplification technique which is overly conservative. Motivated by theoretical interest, we develop new techniques that guarantee exactness, polynomial-time convergence and near-optimal lossless simplifications. We provide an open-source C++ implementation of our variant.

Our primary contribution is an extensive empirical evaluation on a broad, publically available, suite of real-world and synthetic data sets. Among the frog-based variants, exact computation indeed introduces overhead and increases median runtime. Yet, our new approach is often faster in the worst case, worst ten percent, or even the average runtime due to its worst-case convergence guarantees. More surprisingly, the implementation of [SoCG'19] dominates all frog-based implementations in performance – this finding contrasts previously published claims. These results provide a much-needed nuanced perspective on the capabilities and limitations of frog-based techniques: we showcase its theoretical appeal, but highlight its limited practical feasibility.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Algorithms engineering, Fréchet distance

Digital Object Identifier 10.4230/LIPIcs.SoCG.2026.35

Related Version *Full Version*: `https://arxiv.org/abs/2512.07728` [20]

Supplementary Material *Software (Source Code)*: `https://doi.org/10.5281/zenodo.19074030`

Funding *Ivor van der Hoog* and *Eva Rotenberg* are grateful to the Carlsberg Foundation for supporting this research via grant CF21-0302 "Graph Algorithms with Geometric Applications", and to the VILLUM Foundation for supporting this research via grant 37507 "Efficient Recomputations for Changeful Problems". *Jacobus Conradi* is supported by the Carlsberg Foundation, grant CF24-1929.

1 Introduction

The continuous Fréchet distance between two polygonal curves π and σ is classically described using the analogy of a person walking along π and a dog walking along σ, each moving continuously and monotonically. The continuous Fréchet distance $\mathcal{D}_F(\pi, \sigma)$ is the infimum

© Jacobus Conradi, Ivor van der Hoog, and Eva Rotenberg;
licensed under Creative Commons License CC-BY 4.0

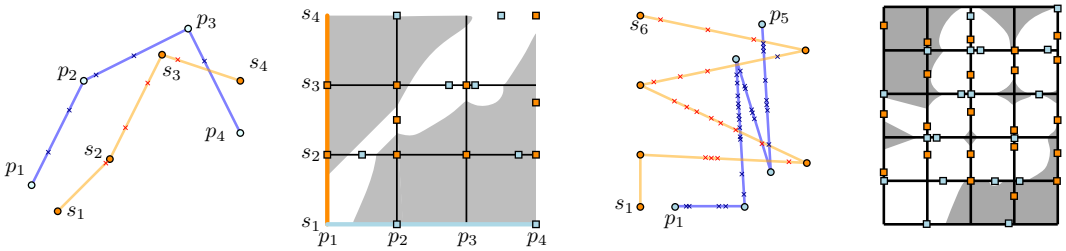

Figure 1 Two curves (π, σ) with their parameter space. White space indicates that the corresponding points are within some distance Δ. In each grid, we illustrate the edge events. For each monotonicity event defined by an edge e and vertices (x_1, x_2) we show the event on e with a cross.

over all such traversals of the maximum distance between them. A discrete analogue is defined similarly by imagining two frogs that hop monotonically along the vertices of π and σ, leading to the discrete Fréchet distance $\mathbb{D}_F(\pi, \sigma)$.

Let $\pi = (p_1, \ldots, p_n)$ and $\sigma = (s_1, \ldots, s_m)$. From an algorithmic and practical perspective, it is possible to decide whether their continuous or discrete decision problems is less than some input value Δ in an efficient manner. For a given value Δ, deciding whether $\mathbb{D}_F(\pi, \sigma) \leq \Delta$ reduces to determining whether there exists a monotone path in an $n \times m$ vertex-weighted grid where the weight of (p_i, s_j) is $d(p_i, s_j)$. Removing all vertices with weight greater than Δ and performing a graph search from (p_1, s_1) to (p_n, s_m) takes $O(nm)$ time. The continuous decision problem admits a related construction that we discuss in the preliminaries.

Computing the Fréchet distance and monotonicity events. Computing the *value* of the Fréchet distance is considerably more intricate. The continuous distance is realised by one of three types of events (see Figure 1). Vertex events correspond to distances between pairs of vertices in $\pi \times \sigma$; edge events correspond to distances between a vertex of one curve and an edge of the other; and monotonicity events correspond to triples (e, s, s'), where e is an edge of one curve, s, s' are vertices of the other curve, and the corresponding distance value is realised by the point on e minimising the maximum distance to s and s'. The discrete distance $\mathbb{D}_F(\pi, \sigma)$ is always realised by a vertex event. As there are $O(nm)$ such events one may therefore compute all distances, sort them, and apply the decision procedure, yielding an $O(nm \log nm)$ algorithm (or $O(nm)$ with a simple dynamic program). This version is also easy to make exact, as one can simply square the vertex-weights to maintain rational input. For the continuous setting, monotonicity events introduce two significant difficulties:

- There are $O(n^3 + m^3)$ such events, so no quadratic-time algorithm can enumerate them.
- Edge and monotonicity events involve equations that contain several square root monomials. They therefore offer no straightforward way to avoid numerical imprecision, even if the input is integral. This causes problems, not only for computing the distance value, but even for computing the correct combinatorial traversal that realises the Fréchet distance.

An exact decision procedure for the continuous variant was provided by [10], who use exact geometric predicates to test $\mathcal{D}_F(\pi, \sigma) \leq \Delta$ in $O(nm)$ time. Their implementation allows an approximation of $\mathcal{D}_F(\pi, \sigma)$ to b bits precision via binary search in $O(nm(\log nm + \log b))$ time, but it does not compute the exact value.

Approaching the continuous Fréchet distance with a frog. [26] recently proposed a different approach. Given (π, σ) they construct what they call the VE-graph whose vertices correspond to all vertex and edge events. In this graph, there is an edge between two vertices if they share

a cell and either the edge is monotone, or the events lie on an edge of the same curve (see Figure 2). The vertices are weighted by the event distance, and the shortest path from $(1, 1)$ to (n, m) in this graph corresponds to a (non-monotone) person-dog traversal γ. If γ happens to be xy-monotone, its continuous counterpart realises the Fréchet distance. Otherwise, non-monotonicities certify that the Fréchet distance is realised by a monotonicity event. Instead of computing monotonicity events, their algorithm bisects each edge of π and σ where γ is non-monotone in the corresponding row or column. This creates higher-complexity curves (π', σ') on which they recurse. They call this *refinement*, and its recursive application yields increasingly complex curves whose vertex and edge events approach the continuous Fréchet distance $\mathcal{D}_F(\pi, \sigma)$. In the limit, this procedure converges to introducing the necessary monotonicity events and thereby computes the continuous Fréchet distance through discrete shortest paths.

Their second contribution is a clever simplification technique that is agnostic to how the Fréchet distance between curves is computed. They show how to simplify two input curves (π, σ) into lower-complexity curves $(\overline{\pi}, \overline{\sigma})$ such that $\mathcal{D}_F(\pi, \sigma) = \mathcal{D}_F(\overline{\pi}, \overline{\sigma})$. The procedure is iterative: they first compute simplified curves $(\overline{\pi}, \overline{\sigma})$ and evaluate $\mathcal{D}_F(\overline{\pi}, \overline{\sigma})$ using any method. This yields a person-dog traversal, from which they generate a witness certifying whether $\mathcal{D}_F(\overline{\pi}, \overline{\sigma}) = \mathcal{D}_F(\pi, \sigma)$ holds. If not, the curves are partially un-simplified, reintroducing complexity until the criterion is satisfied. Based on our observations, this simplification step is likely the main source of the speedy performance of their implementation.

Applications of Fréchet distance. The Fréchet distance has many well-known applications, in particular in the analysis and visualisation of movement data [11, 15, 29, 37], and has enjoyed continued interest in many different settings [1, 2, 7, 17–19, 23]. It is a versatile measure used across a wide range of domains, including handwriting [33], coastlines [30], geographic outlines [21], trajectories of vehicles, animals, and sports players [9, 15, 32, 34], air traffic [8], and protein structures [28]. These typically provide their own Fréchet distance implementations. A recent and particularly active domain is *trajectory clustering*, where the task is to group (sub)curves into clusters such that each cluster has low pairwise Fréchet distance. Several approaches adopt this principle, including the subtrajectory clustering algorithm of [3], the map construction and clustering algorithms [11, 12], the clustering method of [13] (implemented in the `MOVETK` library [24]), the clustering algorithm of [16], and the clustering of [35]. In the *vast* majority of these examples, the implementation computes the discrete Fréchet distance, as this is considerably easier to evaluate in practice.

Two of these approaches [11, 12] compute the *semi-weak* Fréchet distance, in which the person and the dog are not required to be monotone within an edge. This variant corresponds to the edge-monotone minimum-cost traversal computed from [26] before their curve refinement procedure is applied. These examples illustrate the need for efficient and adaptable Fréchet distance implementations, and highlight why the refinement-based approach of [26] is appealing: it suggests the possibility of computing the exact continuous Fréchet distance through a discrete traversal mechanism that is compatible with existing algorithmic paradigms and interfaces.

Contribution. We re-examine the frog-based framework of [26], as we improve three aspects of their contribution.

Contribution 1: exact-value computations. We find the overarching method conceptually very interesting (and even elegant), however, their algorithm does not compute the exact Fréchet distance. They state that "as the calculation of the optimal Fréchet distance involves

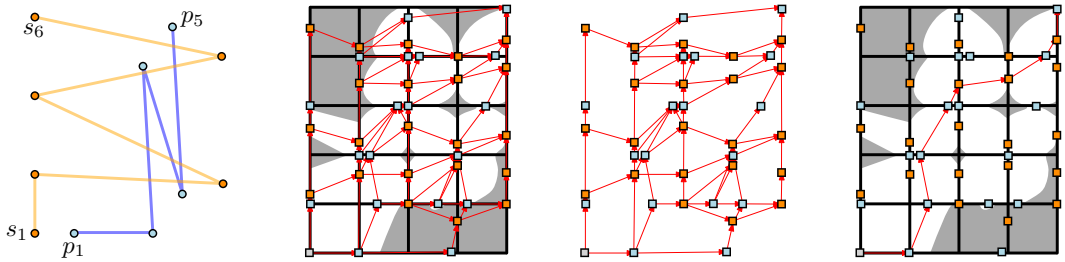

Figure 2 Two pairs of curves (π, σ) and their VE-graph which is a directed and vertex-weighted graph. The minimum-cost path from $(1, 1)$ to (n, m) in this graph is not a monotone traversal.

distances, impreciseness is unavoidable". This is not necessarily true as there exist exact-evaluation predicates. In theory, one can even assume a real RAM. Their current approach would remain imprecise under such assumptions:

1. Their implementation inherently uses floating-point arithmetic and thus imprecision.
2. However, even with exact real-valued arithmetic, their edge bisection does not introduce the monotonicity events that realise the exact Fréchet distance. This refinement converges to the correct value only in the limit, with no bound on the number of required iterations.

Their lossless simplification step is theoretically sound and would yield the exact Fréchet distance under real-valued arithmetic. However, the procedure is somewhat crude: to ensure that the Fréchet distance remains unchanged after simplification, they impose overly conservative constraints on how the curves may be simplified. The original paper briefly acknowledges the possibility of an exact algorithm, but quickly states that "from implementation point of view such a modification seems pointless, and we did not pursue it any further". Our motivation is to understand to what extent this framework *can* be made exact and efficient, and to provide the required components to do so.

1. We explain in detail how to unpack square-root expressions for exact evaluation. This is non-trivial, particularly for monotonicity events and curve simplifications where several square-root terms must be handled simultaneously.
2. We replace their heuristic refinement procedure with a new output-sensitive algorithm that generates the required monotonicity events on demand. As a consequence, we obtain provable polynomial-time convergence, even under real-valued computations.
3. We strengthen the lossless simplification technique. Whilst the method from [26] works as a black-box given any Fréchet computation algorithm that outputs a traversal, we note that this frog-based paradigm allows for a tighter algorithm and analysis. This leads to an alternative and new algorithm for lossless simplification under the frog-based method.

Contribution 2: exact implementation. We provide an open-source C++ implementation of our adapted algorithm with our exact arithmetic and monotonicity event-generation. This thereby is the first exact implementation. We experientially compare our solution to the one provided in [26]. Unsurprisingly, our median running times increase due to exact computation. Our exact-value representations simply do not allow for vectorisation, which comes at performance cost. Somewhat unexpectedly, the worst-case running time drastically improves. We can show instances where the recursive refinement encounters its worst-case behaviour and terminates with an approximation only after many refinement steps. For some data sets, the worst-case behaviour is such that even our average time is an improvement.

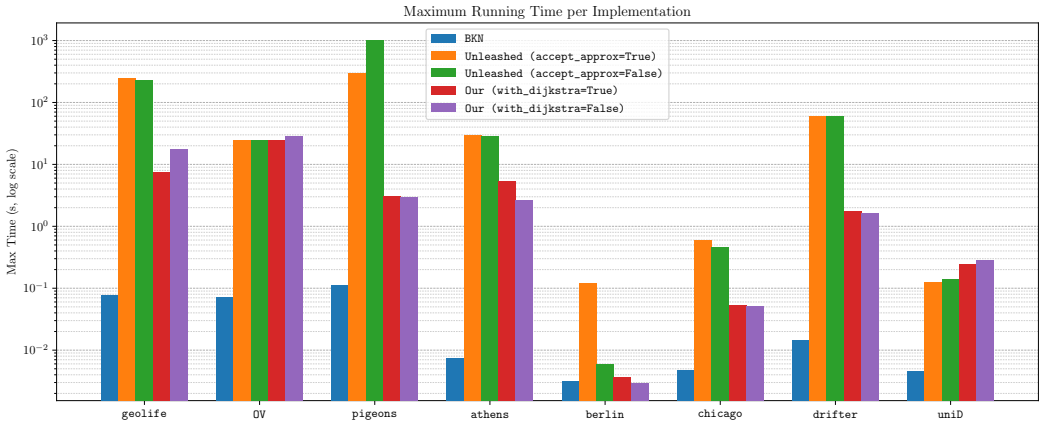

Figure 3 Maximum running times in seconds, log-scale, for curves from eight data sets.

Contribution 3: extensive empirical analysis. Our final contribution is the most significant. The original publication [26] claims that their running times are significantly faster than previous Fréchet computations. Across all our data sets, we observe a substantially different performance profile. In particular, the highly-optimised implementation of Bringmann, Künnemann, and Nusser `BKN` [10] dominates all frog-based implementations in performance (e.g., Figure 3). As a consequence of our discovery, the authors of [26] posted on their ArXiV version an errata explaining that their experimental setup was flawed.

At first sight, our empirical analysis may appear unfavourable to our contribution, as it shows that on many practical instances the performance of this new frog-based approach is surpassed by classical techniques. However, we argue for our significance: we provide the first exact Fréchet distance implementation. Moreover, the empirical analysis reported at SoCG'25 [26] claimed strong practical performance over [10]. Our broader evaluation provides a necessary complementary picture, contradicting this claim. We view this contrast as valuable to our community: it clarifies that this technique is valuable, but also clearly shows the limits of its use. Ultimately, we argue that the frog-based technique of [26] is of genuine theoretical interest, as it can be augmented to compute the exact Fréchet distance in a unique manner with provable guarantees. At the same time, its practical use is debatable.

2 Preliminaries

The input consists of polygonal curves π and σ. A *curve* $\pi = (p_1, \ldots, p_n)$ is an ordered sequence of *vertices* connected by straight-line *edges*. We implicitly endow every n-vertex curve π with a bijective, monotone parametrisation $\pi : [1, n] \to \mathbb{R}^d$ such that $\pi(i) = p_i$ for all integers i. For $t \in [i, i+1]$, the mapping is defined by linear interpolation $\pi(t) := p_i + (p_{i+1} - p_i)\,(t - i)$.

For an ordered pair of curves (π, σ) of sizes (n, m), a *traversal* is any pair of continuous, monotone mappings $\alpha : [0, 1] \to [1, n]$ and $\beta : [0, 1] \to [1, m]$. The (continuous) Fréchet distance $\mathcal{D}_F(\pi, \sigma)$ is defined as $\mathcal{D}_F(\pi, \sigma) := \inf_{\text{traversals } (\alpha,\beta)} \max_{t\in[0,1]} d\big(\pi(\alpha(t)), \sigma(\beta(t))\big)$.

Equivalently, consider the parameter space $\Gamma_{\pi,\sigma} := [1, n] \times [1, m]$, subdivided into an $n \times m$ grid. A traversal corresponds to any continuous, xy-monotone curve $\gamma(t) = (\alpha(t), \beta(t))$ from $(1, 1)$ to (n, m) in $\Gamma_{\pi,\sigma}$. We therefore freely consider traversals as monotone curves.

Deciding. We can decide whether $\mathcal{D}_F(\pi,\sigma) \leq \Delta$ as follows. The area $\{(a,b) \in [1,n]\times[1,m] \mid d(\pi(a),\sigma(b)) \leq \Delta\}$ is the *free space*. Via DFS over the free space we decide in $O(nm)$ time whether the free space contains a monotone curve from $(1,1)$ to (n,m), i.e., $\mathcal{D}_F(\pi,\sigma) \leq \Delta$ [5].

Computing. The Fréchet distance is realised by one of three types of *events*:

- *Vertex events*: distances between pairs of vertices in $\pi \times \sigma$.
- *Edge events*: distances between a vertex of one curve and an edge of the other.
- *Monotonicity events*: triples (e, s, s') where e is an edge of one curve and s, s' are vertices of the other, corresponding to minimising the maximum distance to s and s'.

A naive algorithm enumerates all events in $O(nm^2 + mn^2)$ time, sorts them, and performs a binary search with the decision algorithm to obtain $\mathcal{D}_F(\pi,\sigma)$. [5] observed that many monotonicity events are irrelevant and achieved an $O(nm \log nm)$-time exact algorithm. [25] gave a considerably simpler approach: as Δ increases, connected components of the free space merge, and a *critical value* is the smallest Δ at which a new merge occurs. Each cell has one connected component, implying $O(nm)$ critical values. By comparing adjacent cells, all critical values can be generated in $O(nm \log nm)$ time. The fastest known algorithm is by [14], running in $O\big(n^2 \sqrt{\log n}\,(\log\log n)^{3/2}\big)$ time.

Retractability. For curves (π,σ) and a minimum-cost traversal $\gamma = \{\gamma(t) \mid t \in [0,1]\}$, define $\text{cost}(\gamma) := \max_{(a,b)\in\gamma} d\big(\pi(a),\sigma(b)\big)$. Let $(a,b) \in \gamma$ be a point that attains this maximum. Then γ decomposes into two open curves: γ_1 from $(1,1)$ to (a,b) and γ_2 from (a,b) to (n,m). [26] call γ *retractable* if γ_1 and γ_2 are minimum-cost monotone curves for their respective subcurves, and if γ_1 and γ_2 are themselves retractable (Figure 4). Computing the Fréchet distance using an adaption of Dijkstra yields a retractable traversal.

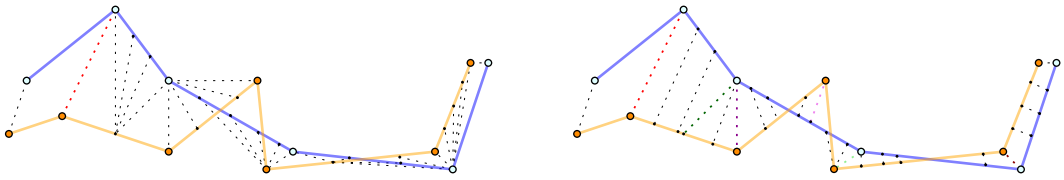

Figure 4 (Left) A traversal between two curves where the red leash realises $\mathcal{D}_F(\pi,\sigma)$. The traversal may be "lazy": a large Fréchet distance allows long pauses while the other curve progresses. (Right) A retractable traversal achieves progressively smaller bottleneck leash lengths.

3 The frog-based approach

We paraphrase and explain the frog-based approach from [26], defining their `VE`-graph slightly differently to reduce the number of edge-cases the definition incurs. We then detail how to make it converge in a polynomial number of rounds, even on a real-RAM, by creating an algorithm that generates the required monotonicity events on the fly.

▶ **Definition 1.** *Each edge e of the grid $\Gamma_{\pi,\sigma}$ has an edge event defining a point that we call an* edge-distance yardstick *(or* eddy*). $V(\pi,\sigma)$ denotes the set of all grid vertices and eddys.*

▶ **Definition 2.** *The `VE`-graph is a directed vertex-weighted graph on $V(\pi,\sigma)$ where the weight of each vertex is the distance between the pairs of points corresponding to the vertex. For each cell of $\Gamma_{\pi,\sigma}$, consider its four corners and eddys. We connect these points in an xy-monotone manner. In addition, the bottom horizontal eddy has a directed edge to the top horizontal eddy, and the left vertical eddy has a directed edge to the right vertical eddy; see Fig. 5.*

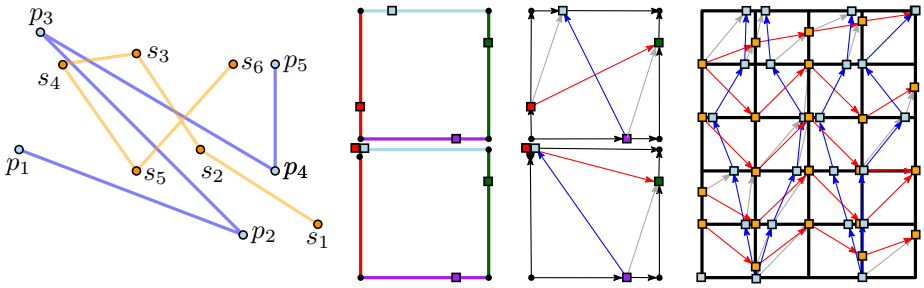

Figure 5 The VE graph considers for each cell the four corners and the four eddys. We include all monotone edges between these vertices. Edges connecting the eddys between horizontal (or vertical) edges may be non-monotone. If vertices coincide, then the picture gets a bit messy.

▶ **Definition 3.** *A path in the VE-graph is VE-respecting if it runs from* $(1,1)$ *to* (n,m) *and, within every row and every column, its restriction to the row or column is connected.*

$$\mathcal{V}_E(\pi,\sigma) := \min_{\text{VE-respecting path } P} \max_{\text{vertex } v \in P} \text{weight}(v). \qquad \text{Observe that } \mathcal{V}_E(\pi,\sigma) \leq \mathcal{D}_F(\pi,\sigma).$$

From a path realizing the VE distance, one may also derive an upper bound on $\mathcal{D}_F(\pi,\sigma)$.

▶ **Definition 4.** *Let* P *be a path realizing* $\mathcal{V}_E(\pi,\sigma)$ *then* P *is a curve. One can greedily morph* P *into a traversal by ensuring that one never decreases in either coordinate. This defines an* xy*-monotone traversal* $\gamma(t) = (\alpha(t), \beta(t))$*. The* Interpolated VE distance *is then defined as:*

$$\mathcal{I}_{VE}(\pi,\sigma) := \max_{t\in[0,1]} d\big(\pi(\alpha(t)), \sigma(\beta(t))\big). \qquad \text{Observe that } \mathcal{D}_F(\pi,\sigma) \leq \mathcal{I}_{VE}(\pi,\sigma).$$

The *elevation function* maps $(a,b) \in \Gamma_{\pi,\sigma}$ to $d(\pi(a),\sigma(b))$. The sublevel sets of this function are clipped ellipses, whose minima on each cell edge occur precisely at the corresponding eddys. It follows that $\mathcal{V}_E(\pi,\sigma) = \mathcal{D}_F(\pi,\sigma) = \mathcal{I}_{VE}(\pi,\sigma)$ whenever the path P realizing $\mathcal{V}_E(\pi,\sigma)$ is monotone. This is the core of the frog-based refinement approach (Algorithm 1).

Algorithm 1 Refinement Procedure from [26].

1: Compute the VE-graph, the value $\mathcal{V}_E(\pi,\sigma)$, and the path P realising it
2: **if** P is monotone **then** $\mathcal{V}_E(\pi,\sigma) = \mathcal{D}_F(\pi,\sigma)$ so **return** $\mathcal{V}_E(\pi,\sigma)$
3: Otherwise, insert vertices on π for every non-x-monotone column, and vertices on σ for every non-y-monotone row. Then recurse on the refined curves (π,σ).

3.1 The approach in [26]: bisecting and convergence

Har-Peled, Raichel, and Robson [26] execute step 3 of Algorithm 1 in a heuristic manner. Let P be a path realising $\mathcal{V}_E(\pi,\sigma)$ for the current input (π,σ). If P is non-monotone in a column that corresponds to an edge e of π then they introduce a vertex half-way e (likewise, they bisect edges of σ for each row where P is non-monotone). Thus, given P they bisect some edges and create higher-complexity curves (π,σ) on which they recurse.

Recall that each monotonicity event (e,p,p') corresponds to a point on the edge e; we refer to such a point as a *monotonicity vertex*. Let (π',σ') be the curves obtained from (π,σ) by inserting, on π and σ respectively, a vertex at every monotonicity vertex. It is observed in [26] that the path P realising $\mathcal{V}_E(\pi',\sigma')$ is xy-monotone, and therefore $\mathcal{V}_E(\pi',\sigma') = \mathcal{D}_F(\pi,\sigma)$. Under exact-value arithmetic, the recursive halving procedure of [26] inserts in the limit all monotonicity vertices of π and σ, and thus converges to the exact Fréchet distance $\mathcal{D}_F(\pi,\sigma)$.

3.2 Our approach and rate of convergence

In [26] they use floating-point arithmetic. Therefore, the computation is non-exact. Moreover, bisecting edges will typically not generate the exact monotonicity vertices. They remark that under exact computations one can compute and introduce monotonicity vertices instead, but do not provide details, and naively there are cubicly many such events. Floating-point arithmetic is not unavoidable. Many libraries support exact arithmetic (e.g., CGAL). We engineer a solution that alters step 3 to provide guarantees for exactness and convergence: Let the path P realising $\mathcal{V}_E(\pi, \sigma)$ be not x-monotone in a column C corresponding to an edge e. Let the subpath of P restricted to C be from a to b. We execute step 3 by computing the minimum number of monotonicity vertices that need to be introduced on e such that, next iteration, the min-cost path from a to b is monotone. Our algorithm is inspired by the critical values from [5, 25] from the preliminaries, but ultimately has a different objective.

On efficiency. Exact-value arithmetic inevitably incurs overhead. Evaluating exact geometric predicates is substantially slower than using floating-point arithmetic. Moreover, the resulting values must be maintained as abstract algebraic objects rather than bit-level representations, preventing efficient vectorisation. This is partly why the algorithm from [10] does not use exact arithmetic even though it is integrated in CGAL. We therefore expect exact computation to increase median running times when we introduce exact arithmetic. To mitigate this effect, we restrict the input to 32-bit integers, as this allows the most frequent operation – the computation of dot products – to still be efficient. For our exact arithmetic, based on the computed dot products, we discuss in the full version our own integer-based kernel, to remain practically somewhat competitive [20]. Importantly, our algorithm does not rely on integrality assumptions: it applies unchanged under any exact-evaluation kernel.

High-level overview. In the full version [20] we develop the following algorithm: the input is a column C where P is not x-monotone, corresponding to an edge e of π. Let P, restricted to C, go from a to b. We create two new cells by extending a horizontal segment from a and b. We discard all cells of C that do not intersect P. We index the remaining cells as $C_1, \cdot, C_k$ from bottom to top, which induce horizontal segments $h_1, \ldots, h_{k+1}$. Imagine Δ increasing from 0 to ∞. For each value of Δ and each pair (i, j) with $i < j$, h_i can Δ-*reach* h_j if there exists a monotone path from a point on h_i to a point on h_j in the Δ free space (see Figure 6).

A value Δ is an *join event* if there exist (i, j) such that h_i can Δ-reach h_j from Δ onward. In this case, h_i can also Δ-reach all $h_{j'}$ with $j' \in [i, j)$. We compute the minimum number of join events to realise a monotone path from h_1 to h_k using a linked–list of buckets.

The horizontal segments are partitioned into buckets, each forming a vertical interval $[i, j]$ such that h_i can Δ-reach h_j. We initialise Δ as 0, and every horizontal segment forms its own bucket. We then merge buckets in a bottom-up fashion, maintaining the minimum Δ required such that h_1 can Δ-reach h_i for increasing i. Between $(h_1, \ldots, h_i)$ and h_{i+1}, we compute the smallest value $\Delta' \geq \Delta'$ for h_1 can Δ-reach h_{i+1} using a min-heap that contains join events and other events that indicate for which pair $(j, i + 1)$ the join event is relevant.

Whenever we pop the next event, we update Δ. If the updated Δ allows us to Δ-reach h_{i+1} from h_1 then we merge h_{i+1} into the bucket. We then keep incrementing i until h_1 cannot Δ-reach h_{i+1}, at which point we continue the event-based approach. Whenever we merge h_k into the bottom bucket, h_1 can Δ-reach h_k. When all buckets are joined, we have all join events required for a monotone path from h_1 to reach h_k. From this, we obtain the minimum number of monotonicity vertices needed so that, in the next iteration, the min-cost path from a to b is monotone. Doing this for all non-monotone rows and columns implies:

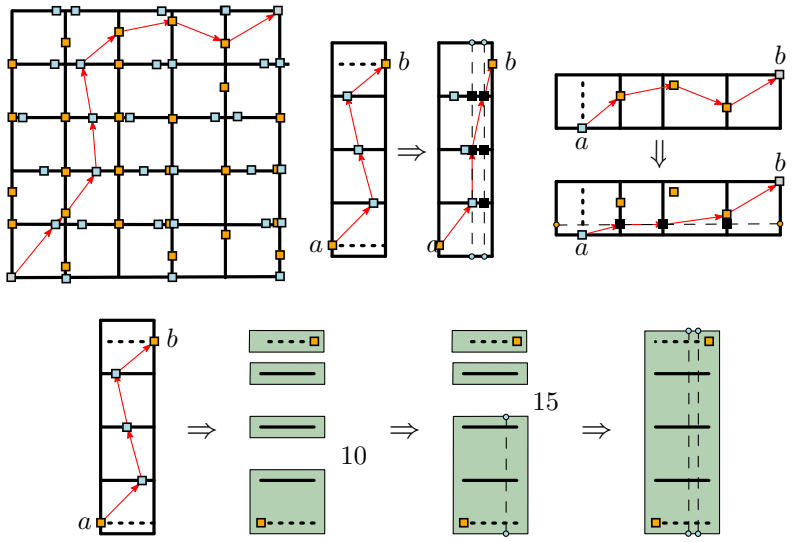

Figure 6 The input is a non-monotone path P in the VE-graph. We identify all rows and columns where P is non-monotone. For a column, we isolate the horizontal segments and bucket them. Between buckets, we store the minimum leash at which a monotone path from bottom to top exists.

▶ **Theorem 5.** *Let (π, σ) be the original input curves with n and m vertices. Algorithm 1 can be supported by a data structure using $O(n + m + M)$ space, where $M \in O(n^3 + m^3)$ is the number of introduced monotonicity vertices. Each recursive call runs in $O(nm + M)$ time for Step 1 and $O((n + m + M) \log^2(n + m + M))$ time for Step 3. Under exact-value arithmetic, the procedure returns the exact Fréchet distance after $O(n^3 + m^3)$ recursive calls.*

4 Lossless Fréchet distance simplification

The second (perhaps even main) contribution of [26] is a black-box lossless simplification algorithm. Computing $\mathcal{D}_F(\pi, \sigma)$ for large complexity curves π and σ is an expensive operation. It would be preferable if we would have access to some lower-complexity curves $(\overline{\pi}, \overline{\sigma})$ where $\mathcal{D}_F(\pi, \sigma) = \mathcal{D}_F(\overline{\pi}, \overline{\sigma})$. Then we could simply operate on these lower-complexity curves to output the distance. Har-Peled, Raichel, and Robson [26] can, for any Fréchet computation algorithm, achieve this using any *vertex-restricted* curve simplification:

▶ **Definition 6.** *We define for π a (vertex-restricted) simplification $\overline{\pi}$ as any subset of vertices in π, ordered by their index. For any $\mu \in \mathbb{R}_{\geq 0}$ we say $\overline{\pi}$ is a μ-simplification if $\mathcal{D}_F(\pi, \overline{\pi}) \leq \mu$.*

The curve simplification used in [26] is some heuristic combination of techniques from [6, 22].

Naïvely, one can $(1 + \varepsilon)$-approximate $\mathcal{D}_F(\pi, \sigma)$ with simplifications by fixing some ε and μ. Compute μ-simplifications $(\overline{\pi}, \overline{\sigma})$ and set $M = \mathcal{D}_F(\overline{\pi}, \overline{\sigma})$. By the triangle inequality, $M - 2\mu \leq \mathcal{D}_F(\pi, \sigma) \leq M + 2\mu$. If $\mu \leq \varepsilon M$, then M $(1 + \varepsilon)$-approximates $\mathcal{D}_F(\pi, \sigma)$; otherwise μ is decreased cleverly until we terminate after $O(\log_{1+\varepsilon} n)$ rounds [22].

[26] avoids this approximation step by exploiting triangle inequalities more directly. Let $(\overline{\pi}, \overline{\sigma})$ be any vertex-restricted simplification of (π, σ) and consider the traversals $\gamma_0(t)$, $\gamma_1(t)$, and $\gamma_2(t)$ realising $\mathcal{D}_F(\pi, \overline{\pi})$, $\mathcal{D}_F(\overline{\pi}, \overline{\sigma})$, and $\mathcal{D}_F(\overline{\sigma}, \sigma)$, respectively. They combine these traversals in linear time into a traversal $\gamma(t) = (\alpha(t), \beta(t))$ between (π, σ) whose maximum

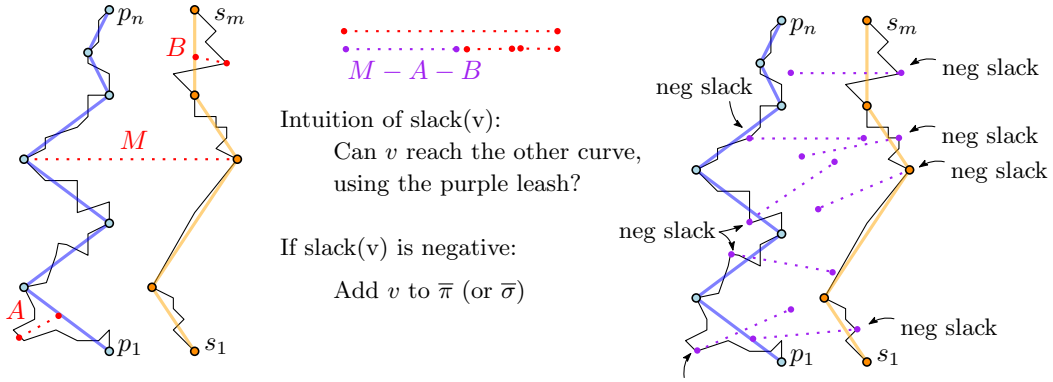

Figure 7 Curves (π, σ) and their simplifications. The value $LB(\overline{\pi}, \overline{\sigma})$ is the maximum leash length M between $(\overline{\pi}, \overline{\sigma})$, minus the maximum leash lengths A and B between $(\pi, \overline{\pi})$ and $(\sigma, \overline{\sigma})$.

leash length gives an upper bound on $\mathcal{D}_F(\pi, \sigma)$. Conversely, $\mathcal{D}_F(\pi, \sigma)$ is at least $\mathcal{D}_F(\overline{\pi}, \overline{\sigma})$ minus the Fréchet distances from each curve to its simplification, and we get the following:

$$\mathcal{D}_F(\overline{\pi}, \overline{\sigma}) - \mathcal{D}_F(\pi, \overline{\pi}) - \mathcal{D}_F(\sigma, \overline{\sigma}) \;\leq\; \mathcal{D}_F(\pi, \sigma) \;\leq\; \max_{t \in [0,1]} d(\pi(\alpha(t)), \sigma(\beta(t))).$$

Computing $\mathcal{D}_F(\pi, \overline{\pi})$ and $\mathcal{D}_F(\sigma, \overline{\sigma})$ is costly, so they compute greedy traversals $\gamma_0^*(t)$ and $\gamma_2^*(t)$ that overestimate these values instead, yielding $\mathcal{D}_F^*(\pi, \overline{\pi})$ and $\mathcal{D}_F^*(\sigma, \overline{\sigma})$. Combined, these give a greedy traversal $\gamma^*(t) = (\alpha^*(t), \beta^*(t))$ that over-estimates $\mathcal{D}_F(\pi, \sigma)$ and leads to

$$LB(\bar{\pi}, \bar{\sigma}) := \mathcal{D}_F(\bar{\pi}, \bar{\sigma}) - \mathcal{D}_F^*(\pi, \bar{\pi}) - \mathcal{D}_F^*(\sigma, \bar{\sigma}) \leq \mathcal{D}_F(\pi, \sigma) \leq \max_{t \in [0,1]} d(\pi(\alpha^*(t)), \sigma(\beta^*(t))). \quad (1)$$

Introducing slack. Given $LB(\overline{\pi}, \overline{\sigma})$ and $\gamma^*(t)$, each curve vertex v receives a *slack* $\texttt{slack}(v) := LB(\overline{\pi}, \overline{\sigma}) - g_v$, where g_v is the maximum leash length attained at v along $\gamma^*(t)$ (Figure 7). Negative slack marks v as a bottleneck. Each vertex p of $\overline{\pi}$ or $\overline{\sigma}$ inherits the minimum slack of all original vertices between p and the next vertex. Slack is then heuristically propagated: every vertex adopts the minimum slack among its eight closest neighbours along $\overline{\pi}$ or $\overline{\sigma}$. The slack of an edge is the minimum slack of its endpoints. They then perform the following:

> while there exists a negative-slack edge e on $\overline{\pi}$ (resp. $\overline{\sigma}$), and the edge e is not an edge of π (resp. σ), un-simplify the edge e (reintroducing missing vertices).

By construction of slack, its propagation, and (1), the process ends with $\mathcal{D}_F(\pi, \sigma) = \mathcal{D}_F(\overline{\pi}, \overline{\sigma})$.

Our contribution. In the full version, we observe that the above technique, while elegant, is overly conservative and somewhat coarse [20]. We refine it in two ways.

(i) Tighter bounds. Their lower bound uses the global maximum leash length M between $(\overline{\pi}, \overline{\sigma})$, minus the global maxima A and B between $(\pi, \overline{\pi})$ and $(\sigma, \overline{\sigma})$, respectively. When M, A and B are realised at unrelated parts of the traversal (see Figure 7), this yields an unnecessarily small value $LB(\overline{\pi}, \overline{\sigma}) = M - A - B$. This in turn causes more vertices than needed to trigger unsimplification. Using the frog-based framework, we obtain a sharper bound (Figure 8). For each edge of $\overline{\pi}$ we compute the maximum leash length between that edge and π, and assign this as a negative weight. In the VE-graph, the eddys on this edge add this negative weight to their own weight. The min-cost path in this reweighted graph

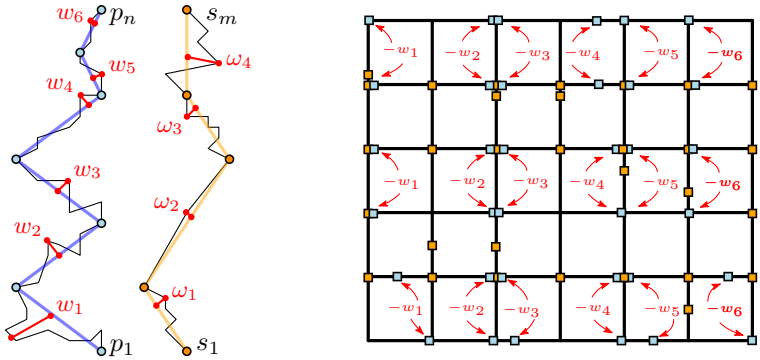

Figure 8 Given $(\pi, \overline{\pi})$ we can give a weight to each edge of $\overline{\pi}$. Each edge over $\overline{\pi}$ is a column in the parameter space, in which we apply its negative weight on the corresponding eddys.

yields a lower bound for $\mathcal{D}_F(\pi, \sigma)$ by the same triangle-inequality argument as in [26], but now based on local rather than global maxima. For many instances, this increases the lower bound and therefore the slack of vertices – causing our algorithm to unsimplify fewer edges.

(ii) A tighter slack propagation rule. Their propagation step takes the minimum slack over each block of eight consecutive vertices. This removes any edge cases for applying the slack, ensuring their correctness. However, this method is also rather crude and causes unnecessary unsimplification. We tighten this step by propagating slack along at most two edges per vertex, avoiding this coarse eight-neighbour spread.

5 Experiments and empirical analysis

Our primary contribution is a C++ implementation of our exact variant of the frog-based technique, together with an extensive empirical evaluation. We provide two implementations: `Exact_Dijkstra`, which computes the path P in the `VE`-graph using Dijkstra's algorithm in a manner similar to [26] and yields a traversal with retractability properties; and `Exact_Sweepline`, which instead employs a sweepline to compute P, guaranteeing linear space usage. We compare our implementations to `Unleashed`: the Julia code of [26], which is their fastest version. To obtain the most accurate output from `Unleashed`, we invoke `frechet_c_compute` with `accept_approximation=false` (this still approximates). We also include a version with approximation parameters enabled. Finally, we include the `BKN` algorithm of [10], the fastest known alternative implementation (which is also inexact). We do not use its CGAL variant, as it is slower, while still being inexact.

Experiments. The analysis in [26] uses three animal-tracking data sets: `Geolife`, `Birds`, and `Pigeons`. These trajectories are goal-oriented. Their implementation benefits from this in one major way compared to [10]: as a simplification is applied, this greatly reduces complexity. Consequently, they operate on significantly smaller effective input.

The lossless simplification of [26] applies to any Fréchet algorithm, including [10]. We *could* enable a direct comparison by applying it to [10]. Yet, we deliberately refrain from doing so: the simplification incurs overhead, and [10] is so fast that this may actually slow it down. Despite not using simplifications, our experiments will show that [10] dominates the frog-based approaches, removing the need to improve it further. All experiments are single-threaded and executed on an AMD Ryzen 5 7600X (4.7 GHz) with 96 GB RAM.

Table 1 Overview of all data sets used in our experiments.

Name	Real world	# curves	total # vertices	median # vertices	# of curves compared
`Geolife`	yes	18671	24895648	506	300 choose 2
`Pigeons`	yes	539	6229319	7502	300 choose 2
`Athens`	yes	120	72439	662	120 choose 2
OV	no	220	39060	130	110 (already pairs)
`Chicago`	yes	888	118360	121	100 choose 2
`Berlin`	yes	27188	192223	7	100 choose 2
`Drifter`	yes	2012	1792084	997	100 choose 2
`UniD`	yes	362	214076	495	100 choose 2

Data. Our real-world data sets are summarised in Table 1. The first two (`Geolife`, `Pigeons`) are from [26]. We additionally include `Athens`, `Chicago`, and `Berlin` GPS traffic data sets, widely used in Fréchet benchmarks [4, 11, 27, 36]. We include the `Drifter` data set of [16], containing drifting-buoy trajectories. We add the UnID data set from [31] which is a traffic dataset captured on German highways. Except for `Drifter`, these sets are goal-oriented with minimal backtracking; lossless simplification therefore removes relatively many vertices, which favours the frog-based approaches. Finally, we include a set `OV`[1] of difficult synthetic pairs of curves that are derived from Orthogonal Vectors instances [10].

The `Geolife` data set alone induces over 170,000,000 pairs of curves, so we cannot possibly compute the Fréchet distance between all pairs of curves. We follow [26] and subsample. In our case, we compare all 45,000 pairs among a subselection of 300 curves from the `Geolife` and `Pigeons` data sets. We further augment the analysis by comparing all curves from the `Athens` data set and from the adversarial `OV` data set. To broaden our evaluation, we also include the remaining real-world data sets, where, due to time constraints, we subsample curves and compare all pairs among them. The experiments are primarily bottlenecked by the `Unleashed` implementation[2], and the full evaluation requires several days to complete. Our results indicate no substantial qualitative differences between sampling 100×100 or 300×300 curves from the same data.

Criteria. Our primary evaluation metric is running time (including file-reading time). For exactness, our approach is the most accurate if the input coordinates fit within a 32-bit integer, followed closely by `BKN`; Table 2 briefly reports accuracies for the data sets satisfying this condition. We could compare across all data by truncating all input to 32-bit coordinates, but we refrain from doing so since `BKN` can handle this data natively (albeit inexact).

5.1 Comparing frog-based approaches

We first compare our frog-based implementations to `Unleashed`. This comparison is motivated mainly by scientific curiosity: exact-value computation is more difficult than computing an approximation, so one naturally expects a performance penalty. Our interest is in quantifying this penalty – in other words, in understanding the practical cost

[1] Obtained via personal communication with one author from [10], available with our source code
[2] A time limit of 5 minutes was hit multiple times

Table 2 Accuracy errors, obtained by converting exact square root distances to a `double`.

Data set	Max	Mean	Median
		`Exact`	
`OV`	0	0	0
`athens`	0	0	0
`UnID`	0	0	0
		`BKN`	
`OV`	$3.039939 \cdot 10^{-13}$	$2.756980 \cdot 10^{-13}$	$3.039939 \cdot 10^{-13}$
`athens`	$9.337306 \cdot 10^{-10}$	$5.865283 \cdot 10^{-12}$	$1.269745 \cdot 10^{-12}$
`UnID`	$9.337306 \cdot 10^{-10}$	$5.865283 \cdot 10^{-12}$	$1.269745 \cdot 10^{-12}$
		`Unleashed`	
`OV`	$4.499666 \cdot 10^{-6}$	$1.430129 \cdot 10^{-6}$	$1.181895 \cdot 10^{-16}$
`athens`	$9.267924 \cdot 10^{-6}$	$1.948025 \cdot 10^{-7}$	$9.974841 \cdot 10^{-15}$
`UnID`	$9.337306 \cdot 10^{-10}$	$5.865283 \cdot 10^{-12}$	$1.269745 \cdot 10^{-12}$

of exactness. Table 3 compares the efficacy of the `Unleashed` implementation versus `Exact_Sweepline` (the comparison remains roughly the same when we set for `Unleashed` the boolean `accept_approximation=true`, or when we compare to `Exact_Dijkstra` instead).

As expected, the median running time of our exact variants deteriorates. However, we note that our exact approach has considerably better worst-case performance, even on this advantageous real-world data. On some instances, `Unleashed` is a factor 100 slower. Whenever our worst-case is slower, it is not slower by much. For data sets where curves have sizable complexity, our method becomes the preferred method from the 90th percentile onwards. On the mapconstruction data sets, the median number of vertices is very small. The OV data set is a difficult instance where our exact computations are extremely costly. We note that except for worst-case outliers, the approaches are relatively competitive. We think that this data nicely represent the cost of exact value computation, and the benefit of worst-case guarantees for runtime.

5.2 Comparing Fréchet distance computations

We now provide a comparison between our exact frog-based method, the approach of [26], and the state-of-the-art `BKN` implementation [10]. In this wider evaluation, we consider two configurations of `Unleashed`: the version with `accept_approximation=false` (which, still returns an approximation) and the supposedly faster configuration that sets `accept_approximation=true`. Our exact implementations use either Dijkstra or a sweepline.

5.3 Results

Table 4 shows the full set of experimental results. These results highlight the volatility of the frog-based approach: the standard deviation of these approaches is considerably larger than `BKN` (although lessened for our implementation). Regardless of evaluation criterion (maximum, mean or median running time) the `BKN` algorithm dominates in performance[3].

[3] Due to technical constraints, there is system overhead negatively influencing the measured time for `BKN`. This is particularly noticeable for instances of the `berlin` data set since they have so few vertices. Their approach is the fastest overall in our measurements, but it is arguably still faster.

Table 3 Summary of running times in seconds. For both `Unleashed` (`accept_approx = false`) and `Exact_Sweeline`. The best times across the two methods are in blue.

Data set	Max	Mean	StdDev	Median	80th %ile	90th %ile	95th %ile	99th %ile
				`Unleashed (accept_approx=false)`				
geolife	233.6160	0.3673	3.7389	**0.0069**	**0.0621**	0.3263	1.0108	6.4489
pigeons	>300	0.8508	9.0841	**0.0222**	**0.1645**	0.9395	3.7772	13.5008
athens	29.1741	0.2133	1.3476	**0.0190**	**0.1091**	**0.2808**	**0.5125**	3.6473
OV	**24.6879**	**1.2132**	3.2387	**0.0580**	**1.1886**	**3.7313**	**5.2031**	**15.0121**
chicago	0.4679	**0.0036**	0.0147	**0.0024**	**0.0040**	**0.0053**	**0.0063**	**0.0088**
berlin	0.0059	**0.0001**	0.0001	**0.0001**	**0.0001**	**0.0001**	**0.0001**	**0.0002**
uniD	**0.1402**	**0.0023**	0.0048	**0.0019**	**0.0027**	**0.0032**	**0.0041**	**0.0076**
drifter	61.2818	0.1199	1.1781	**0.0091**	**0.0555**	0.1919	0.4474	1.9376
				`Exact (with_dijkstra=false)`				
geolife	**17.5543**	**0.0984**	0.1903	0.0505	0.1330	**0.2312**	**0.3468**	**0.7444**
pigeons	**2.9708**	**0.1649**	0.1472	0.1265	0.2305	**0.3153**	**0.4130**	**0.7262**
athens	**2.6424**	**0.1617**	0.2359	0.0851	0.2136	0.3816	0.5906	**1.2356**
OV	28.2322	4.5376	6.0343	1.5410	8.5852	13.5812	16.9130	23.8446
chicago	**0.0522**	0.0075	0.0073	0.0043	0.0124	0.0179	0.0231	0.0326
berlin	**0.0029**	0.0004	0.0002	0.0003	0.0004	0.0005	0.0007	0.0014
uniD	0.2809	0.0077	0.0102	0.0064	0.0096	0.0114	0.0139	0.0266
drifter	**1.6604**	**0.0743**	0.0889	0.0505	0.1123	**0.1609**	**0.2215**	**0.4069**

We illustrate the performance profile of these algorithms using Dolan–Moré plots (Figures 9–12). These plots can be read as follows. The x-axis shows a factor $\tau \geq 1$, indicating how many times slower an algorithm may be compared to the best on each instance. The y-axis shows the percentage of instances on which this holds. Thus, if an algorithm's plot contains the point (p, τ) then on p percent of the instances, it is τ times slower than the best performing algorithm. Similarly, $\tau = 1$ reveals how often each algorithm is the best, while a horizontal line such as $p = 90$ indicates how much slower the worst 10% of cases become (which, for `BKN`, is minimal). As τ increases, the curves show how quickly each algorithm "catches up". An algorithm is considered better if its curve approaches the horizontal line $p = 100$ more rapidly. These profiles mirror our observations: `BKN` is consistent; our method lags on median instances, however if the data set contains high enough complexity curves, it frequently and significantly overtakes `Unleashed` as we approach the worst case.

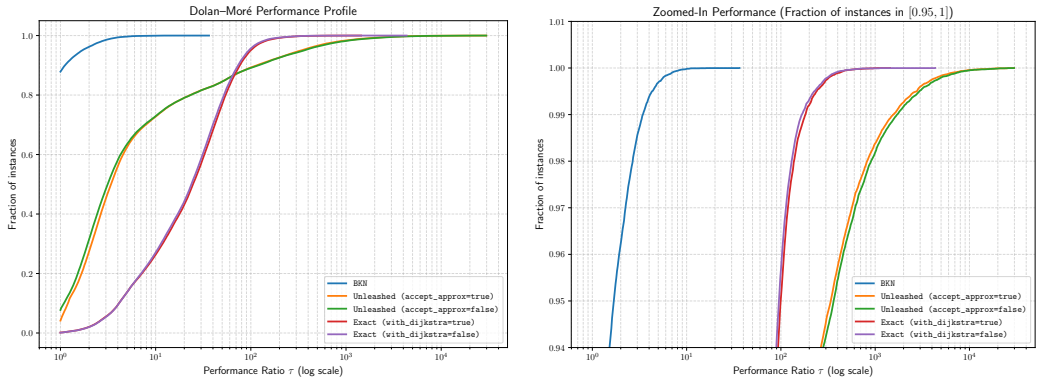

Figure 9 Dolan–Moré performance profile on all pairs of the first 300 curves from the `geolife` data set (44850 in total) from [26].

Table 4 Running time statistics (s). The best running time in bold, the second best in blue.

Data set	Max	Mean	StdDev	Median	80th %ile	90th %ile	95th %ile	99th %ile
				BKN				
geolife	**0.0767**	**0.0024**	0.0016	**0.0021**	**0.0027**	**0.0034**	**0.0042**	**0.0070**
pigeons	**0.1143**	**0.0040**	0.0022	**0.0035**	**0.0048**	**0.0066**	**0.0076**	**0.0102**
athens	**0.0075**	**0.0027**	0.0008	**0.0025**	**0.0033**	**0.0038**	**0.0043**	**0.0055**
OV	**0.0728**	**0.0208**	0.0206	**0.0111**	**0.0376**	**0.0580**	**0.0629**	**0.0682**
chicago	**0.0047**	**0.0018**	0.0002	**0.0018**	**0.0019**	**0.0020**	**0.0021**	**0.0024**
berlin	**0.0032**	0.0016	0.0002	0.0015	0.0017	0.0018	0.0018	0.0019
uniD	**0.0046**	**0.0018**	0.0002	**0.0018**	**0.0020**	**0.0021**	**0.0022**	**0.0024**
drifter	**0.0144**	**0.0024**	0.0009	**0.0022**	**0.0027**	**0.0032**	**0.0037**	**0.0060**
				Unleashed (accept_approx=true)				
geolife	250.5627	0.3367	3.5109	0.0074	0.0623	0.3244	0.9207	5.8376
pigeons	274.5574	0.7529	4.6066	0.0225	**0.1623**	0.9021	3.7495	13.2745
athens	29.4879	0.1754	1.1123	0.0195	**0.1081**	0.2415	0.4817	2.4792
OV	24.8995	1.2463	3.2985	0.0609	1.3038	3.8347	5.5689	15.1980
chicago	0.5972	**0.0035**	0.0144	**0.0023**	**0.0040**	0.0054	0.0064	0.0093
berlin	0.1214	**0.0001**	0.0018	**0.0001**	**0.0001**	**0.0001**	**0.0002**	**0.0002**
uniD	**0.1239**	0.0024	0.0045	0.0020	0.0029	0.0035	0.0042	0.0086
drifter	60.7184	0.1177	1.1667	0.0095	**0.0529**	0.1730	0.4747	1.8644
				Unleashed (accept_approx=false)				
geolife	233.6160	0.3673	3.7389	**0.0069**	**0.0621**	0.3263	1.0108	6.4489
pigeons	>300	0.8508	9.0841	**0.0222**	0.1645	0.9395	3.7772	13.5008
athens	29.1741	0.2133	1.3476	**0.0190**	0.1091	0.2808	0.5125	3.6473
OV	**24.6879**	**1.2132**	3.2387	**0.0580**	**1.1886**	**3.7313**	**5.2031**	**15.0121**
chicago	0.4679	0.0036	0.0147	0.0024	0.0040	**0.0053**	**0.0063**	**0.0088**
berlin	0.0059	**0.0001**	0.0001	**0.0001**	**0.0001**	**0.0001**	**0.0001**	**0.0002**
uniD	0.1402	**0.0023**	0.0048	**0.0019**	**0.0027**	**0.0032**	**0.0041**	**0.0076**
drifter	61.2818	0.1199	1.1781	**0.0091**	0.0555	0.1919	0.4474	1.9376
				Exact (with_dijkstra=true)				
geolife	**7.6259**	0.1028	0.1874	0.0525	0.1381	0.2414	0.3633	0.7742
pigeons	3.1337	0.1672	0.1563	0.1268	0.2322	0.3201	0.4239	0.7432
athens	5.3387	0.1975	0.3426	0.0912	0.2434	0.4529	0.7634	1.6910
OV	24.4039	3.3321	5.0168	0.7700	6.7386	10.2982	14.0325	19.9612
chicago	0.0543	0.0076	0.0076	0.0043	0.0122	0.0182	0.0241	0.0344
berlin	0.0037	**0.0004**	0.0002	**0.0003**	**0.0004**	**0.0004**	0.0006	**0.0014**
uniD	0.2445	0.0077	0.0098	0.0064	0.0095	0.0114	0.0138	0.0271
drifter	1.7855	0.0806	0.1061	0.0519	0.1182	0.1748	0.2428	0.4936
				Exact (with_dijkstra=false)				
geolife	17.5543	**0.0984**	0.1903	0.0505	0.1330	**0.2312**	**0.3468**	**0.7444**
pigeons	**2.9708**	**0.1649**	0.1472	0.1265	0.2305	**0.3153**	**0.4130**	**0.7262**
athens	**2.6424**	**0.1617**	0.2359	0.0851	0.2136	**0.3816**	**0.5906**	**1.2356**
OV	28.2322	4.5376	6.0343	1.5410	8.5852	13.5812	16.9130	23.8446
chicago	**0.0522**	0.0075	0.0073	0.0043	0.0124	0.0179	0.0231	0.0326
berlin	**0.0029**	**0.0004**	0.0002	**0.0003**	**0.0004**	0.0005	0.0007	**0.0014**
uniD	0.2809	0.0077	0.0102	0.0064	0.0096	0.0114	0.0139	0.0266
drifter	**1.6604**	**0.0743**	0.0889	0.0505	0.1123	**0.1609**	**0.2215**	**0.4069**

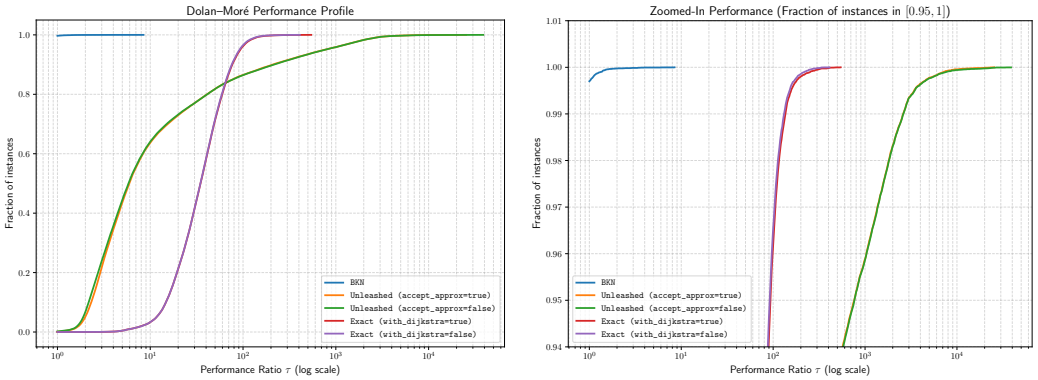

Figure 10 Dolan–Moré performance profile on all pairs of the first 300 curves from the `pigeons` data set (44850 in total) from [26].

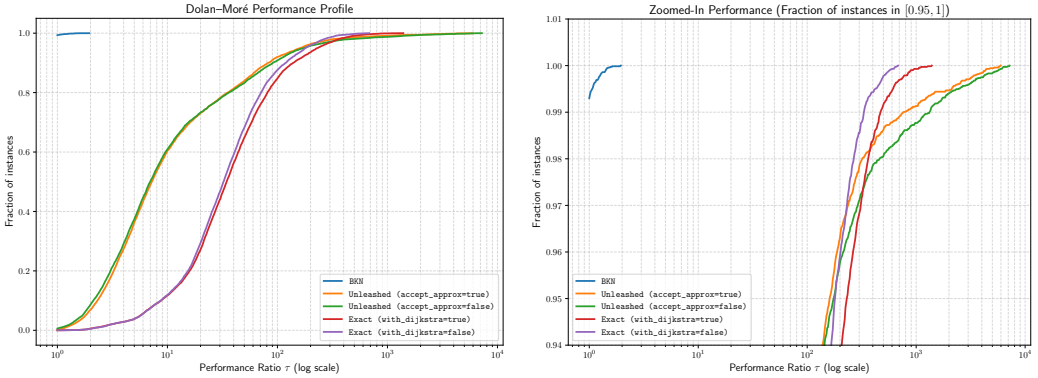

Figure 11 Dolan–Moré profile on pairs from the `athens` data set (7139 in total) from [35].

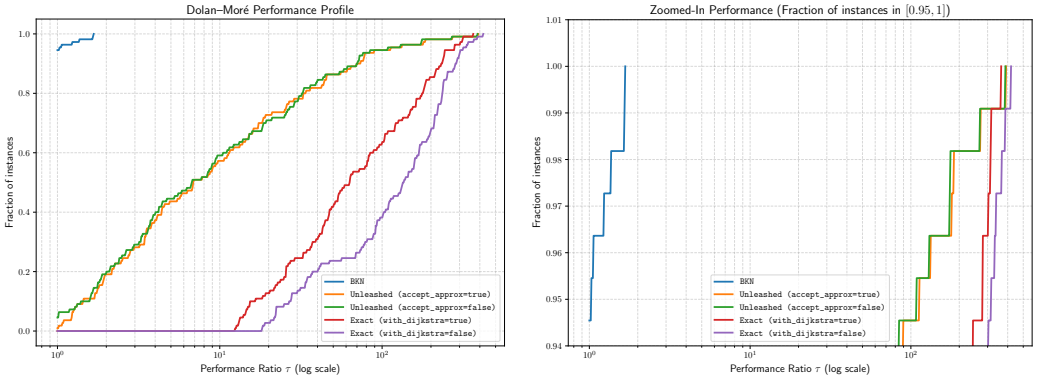

Figure 12 Dolan–Moré performance profile on all pairs from the `OV` data set (110 in total).

6 Conclusion

We studied the frog-based framework for computing the Fréchet distance introduced in [26]. On the theoretical side, we identified three aspects of the original algorithm that incur avoidable inaccuracies in the distance computation or introduce avoidable heuristic overhead.

Specifically, the method of [26] consists of two main components. The first is the computation of the Fréchet distance between (π, σ) via a discrete weighted graph whose complexity is increased iteratively each round, through a process they call *refinement*. Their process heuristically bisects edges of (π, σ), increasing the graph complexity, until the discrete min-cost path through this graph realises the Fréchet distance. We provide a new refinement scheme that offers worst-case guarantees on this process. Our more elaborate refinement is careful not to drastically increase the time per round, yet it yields an explicit upper bound on the convergence rate – which we consider theoretically meaningful.

The second component is a lossless curve simplification procedure that produces curves $(\overline{\pi}, \overline{\sigma})$ of lower complexity such that $\mathcal{D}_F(\pi, \sigma) = \mathcal{D}_F(\overline{\pi}, \overline{\sigma})$ – effectively reducing the input size. Although correct, the original simplification is overly conservative: it relies on a lower bound that is unnecessarily weak, and it avoids geometric corner cases by aggressively propagating vertex slack across a constant but wide neighbourhood. Exploiting the weighted-graph structure already inherent in the frog-based method, we present a more refined construction that attains a higher lower bound. Consequently, our simplifications have lower complexity.

Our primary contribution is an empirical evaluation. We supply an openly available test suite, which enables a reproducible comparison between frog-based implementations and the highly optimised algorithm of Bringmann, Künnemann and Nusser [10]. Our results stand in contrast to the conclusions of [26]: On our broader range of real-world data sets, `BKN` consistently outperforms both frog-based variants. Notably, after removing near-duplicate consecutive points through simplification, the data set curve have *very* few vertices remaining for modern hardware. We have reason to believe that, on larger inputs, the performance gap between our method and `Unleashed` widens. However, we are not aware of real-world benchmark sets whose median complexity exceeds 1,000 "distinct" vertices, and synthetic data behaves entirely differently under lossless simplification. We argue that existing benchmarks are becoming outdated and that the community should develop more modern ones.

We note that the `BKN` algorithm is currently very fast, but still inexact (although it obtains very good approximation ratios). We find it an interesting open problem to augment this code base to allow for exact computations. We note that this is not as straightforward as adding an exact computation kernel (which, the codebase variant in CGAL already has) but that it also requires algorithmic changes to change its decision variant algorithm into the actual computation of the Fréchet distance.

Regarding the experimental results, our findings might seem discouraging: our implementation and analysis describe an approach that does not, in practice, match the performance of the current state-of-the-art. However, we consider this outcome important for the community. Our study offers a transparent and verifiable assessment of the frog-based method and demonstrates that its practical behaviour is volatile and underwhelming.

At the same time, our analysis reaffirms that the frog-based framework is theoretically compelling. We show that this method in particular enables a significantly more fine-grained lossless simplification procedure than general-purpose methods. Nevertheless, our experiments indicate that its main value lies in its theoretical insights rather than in practical performance.

References

1 Peyman Afshani and Anne Driemel. On the complexity of range searching among curves. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, pages 898–917, USA, 2018. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611975031.58.

2 Pankaj K. Agarwal, Rinat Ben Avraham, Haim Kaplan, and Micha Sharir. Computing the Discrete Fréchet Distance in Subquadratic Time. *SIAM Journal on Computing*, 43(2):429–449, 2014. doi:10.1137/130920526.

3 Pankaj K Agarwal, Kyle Fox, Kamesh Munagala, Abhinandan Nath, Jiangwei Pan, and Erin Taylor. Subtrajectory clustering: Models and algorithms. *ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*, pages 75–87, 2018. doi:10.1145/3196959.3196972.

4 Mahmuda Ahmed, Sophia Karagiorgou, Dieter Pfoser, and Carola Wenk. A comparison and evaluation of map construction algorithms using vehicle tracking data. *GeoInformatica*, 19:601–632, 2015. doi:10.1007/s10707-014-0222-6.

5 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5:75–91, 1995. doi:10.1142/S0218195995000064.

6 Boris Aronov, Sariel Har-Peled, Christian Knauer, Yusu Wang, and Carola Wenk. Fréchet Distance for Curves, Revisited. In *Proceedings of the 14th Annual European Symposium on Algorithms (ESA)*, volume 4168 of *Lecture Notes in Computer Science*, pages 52–63. Springer, 2006. doi:10.1007/11841036_8.

7 Lotte Blank, Jacobus Conradi, Anne Driemel, Benedikt Kolbe, André Nusser, and Marena Richter. Transforming Dogs on the Line: On the Fréchet Distance Under Translation or Scaling in 1D. In Oswin Aichholzer and Haitao Wang, editors, *41st International Symposium on Computational Geometry (SoCG 2025)*, volume 332 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:16, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SoCG.2025.22.

8 Alessandro Bombelli, Lluis Soler, Eric Trumbauer, and Kenneth D Mease. Strategic Air Traffic Planning with Fréchet distance aggregation and rerouting. *Journal of Guidance, Control, and Dynamics*, 40(5):1117–1129, 2017. doi:10.2514/1.G002308.

9 Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk. On map-matching vehicle tracking data. In *International Conference on Very Large Data Bases (VLDB)*, pages 853–864, 2005. doi:10.5555/1083592.1083691.

10 Karl Bringmann, Marvin Künnemann, and André Nusser. Walking the Dog Fast in Practice: Algorithm Engineering of the Fréchet Distance. In *Symposium on Computational Geometry (SoCG)*, volume 129 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:21, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SOCG.2019.17.

11 Kevin Buchin, Maike Buchin, David Duran, Brittany Terese Fasy, Roel Jacobs, Vera Sacristan, Rodrigo I Silveira, Frank Staals, and Carola Wenk. Clustering trajectories for map construction. *ACM International Conference on Advances in Geographic Information Systems (SIGSPATIAL)*, pages 1–10, 2017. doi:10.1145/3139958.3139964.

12 Kevin Buchin, Maike Buchin, Joachim Gudmundsson, Jorren Hendriks, Erfan Hosseini Sereshgi, Vera Sacristán, Rodrigo I Silveira, Jorrick Sleijster, Frank Staals, and Carola Wenk. Improved map construction using subtrajectory clustering. *ACM Workshop on Location-Based Recommendations, Geosocial Networks, and Geoadvertising (SIGSPATIAL)*, pages 1–4, 2020. doi:10.1145/3423334.3431451.

13 Kevin Buchin, Maike Buchin, Joachim Gudmundsson, Maarten Löffler, and Jun Luo. Detecting commuting patterns by clustering subtrajectories. *International Journal of Computational Geometry & Applications*, 21(03):253–282, 2011. doi:10.1142/S0218195911003638.

14 Kevin Buchin, Maike Buchin, Wouter Meulemans, and Wolfgang Mulzer. *Four Soviets Walk the Dog—with an Application to Alt's Conjecture*, pages 1399–1413. SIAM, 2014. doi:10.1137/1.9781611973402.103.

15 Maike Buchin, Bernhard Kilgus, and Andrea Kölzsch. Group diagrams for representing trajectories. *International Journal of Geographical Information Science*, 34(12):2401–2433, 2020. doi:10.1145/3283207.3283208.

16 Jacobus Conradi and Anne Driemel. Finding complex patterns in trajectory data via geometric set cover. *arXiv preprint arXiv:2308.14865*, 2023. doi:10.48550/arXiv.2308.14865.

17 Jacobus Conradi and Anne Driemel. On Computing the k-Shortcut Fréchet Distance. *ACM Trans. Algorithms*, 20(4), August 2024. doi:10.1145/3663762.

18 Jacobus Conradi, Anne Driemel, and Benedikt Kolbe. Revisiting the Fréchet distance between piecewise smooth curves. *Computational Geometry*, 129:102194, 2025. doi:10.1016/j.comgeo.2025.102194.

19 Jacobus Conradi, Benedikt Kolbe, Ioannis Psarros, and Dennis Rohde. Fast approximations and coresets for (k,ℓ)-median under dynamic time warping. In Wolfgang Mulzer and Jeff M. Phillips, editors, *40th International Symposium on Computational Geometry, SoCG 2024, Athens, Greece, June 11-14, 2024*, volume 293 of *LIPIcs*, pages 42:1–42:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.SOCG.2024.42.

20 Jacobus Conradi, Ivor van der Hoog, and Eva Rotenberg. On computing the (exact) Fréchet distance with a frog, 2025. arXiv:2512.07728.

21 Thomas Devogele. A new merging process for data integration based on the discrete Fréchet distance. In *Advances in spatial data handling*, pages 167–181, 2002. doi:10.1007/978-3-642-56094-1_13.

22 Anne Driemel, Sariel Har-Peled, and Carola Wenk. Approximating the Fréchet distance for realistic curves in near linear time. In *International Symposium on Computational Geometry (SoCG)*, pages 365–374. ACM, 2010. doi:10.1145/1810959.1811019.

23 Anne Driemel, Ivor van der Hoog, and Eva Rotenberg. On the Discrete Fréchet Distance in a Graph. In Xavier Goaoc and Michael Kerber, editors, *38th International Symposium on Computational Geometry (SoCG 2022)*, volume 224 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 36:1–36:18, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SoCG.2022.36.

24 University of Technology Eindhoven. MoveTK: the movement toolkit. https://github.com/movetk/movetk, 2020.

25 Sariel Har-Peled and Benjamin Raichel. The Fréchet distance revisited and extended. In *Symposium on Computational Geometry (SoCG)*, pages 448–457, New York, NY, USA, 2011. Association for Computing Machinery. doi:10.1145/1998196.1998269.

26 Sariel Har-Peled, Benjamin Raichel, and Eliot W. Robson. The Fréchet Distance Unleashed: Approximating a Dog with a Frog. In Oswin Aichholzer and Haitao Wang, editors, *International Symposium on Computational Geometry (SoCG)*, volume 332 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 54:1–54:13, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SoCG.2025.54.

27 Jincai Huang, Min Deng, Jianbo Tang, Shuling Hu, Huimin Liu, Sembeto Wariyo, and Jinqiang He. Automatic generation of road maps from low quality GPS trajectory data via structure learning. *IEEE Access*, 6:71965–71975, 2018. doi:10.1109/ACCESS.2018.2882581.

28 Minghui Jiang, Ying Xu, and Binhai Zhu. Protein structure–structure alignment with discrete Fréchet distance. *Journal of bioinformatics and computational biology*, 6(01):51–64, 2008. doi:10.1142/s0219720008003278.

29 Maximilian Konzack, Thomas McKetterick, Tim Ophelders, Maike Buchin, Luca Giuggioli, Jed Long, Trisalyn Nelson, Michel A Westenberg, and Kevin Buchin. Visual analytics of delays and interaction in movement data. *International Journal of Geographical Information Science*, 31(2):320–345, 2017. doi:10.5555/3048386.3048392.

30 Ariane Mascret, Thomas Devogele, Iwan Le Berre, and Alain Hénaff. Coastline matching process based on the discrete Fréchet distance. In *Progress in Spatial Data Handling*, pages 383–400. Springer, 2006. doi:10.1007/3-540-35589-8_25.

31 Tobias Moers, Lennart Vater, Robert Krajewski, Julian Bock, Adrian Zlocki, and Lutz Eckstein. The exiD Dataset: A Real-World Trajectory Dataset of Highly Interactive Highway Scenarios in Germany. *2022 IEEE Intelligent Vehicles Symposium (IV)*, pages 958–964, 2022. doi:10.1109/IV51971.2022.9827305.

32 Roniel Sousa, Azzedine Boukerche, and Antonio Loureiro. Vehicle Trajectory Similarity: Models, Methods, and Applications. *ACM Computing Surveys*, 53(5), 2020. doi:10.1145/3406096.

33 E Sriraghavendra, K Karthik, and Chiranjib Bhattacharyya. Fréchet distance based approach for searching online handwritten documents. In *Ninth International Conference on Document Analysis and Recognition (ICDAR)*, pages 461–465, 2007. doi:10.1109/ICDAR.2007.4378752.

34 Han Su, Shuncheng Liu, Bolong Zheng, Xiaofang Zhou, and Kai Zheng. A survey of trajectory distance measures and performance evaluation. *Proceedings of the VLDB Endowment*, 29(1):3–32, 2020. doi:10.1007/s00778-019-00574-9.

35 Ivor van der Hoog, Lara Ost, Eva Rotenberg, and Daniel Rutschmann. Efficient Greedy Discrete Subtrajectory Clustering. In *International Symposium on Computational Geometry (SoCG)*, volume 332 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 78:1–78:20, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SOCG.2025.78.

36 Suyi Wang, Yusu Wang, and Yanjie Li. Efficient map reconstruction and augmentation via topological methods. *ACM International Conference on Advances in Geographic Information Systems (SIGSPATIAL)*, pages 1–10, 2015. doi:10.1145/2820783.2820833.

37 Dong Xie, Feifei Li, and Jeff M Phillips. Distributed trajectory similarity search. *Proceedings of the VLDB Endowment*, 10(11):1478–1489, 2017. doi:10.14778/3137628.3137655.