technische universität
dortmund

Seminar Algorithm Engineering 22/23

Fréchet Distance

Tom Stein

December 05, 2022

Supervisor:
Dr. Carolin Rehs

Technische Universität Dortmund
Department of Computer Science
Chair 11 (Algorithm Engineering)
https://ls11-www.cs.tu-dortmund.de/

Abstract

The Fréchet distance is a metric used to compare curves in a metric space. It measures the similarity between two curves based on the overall shape and course of the curves. This report will provide an overview of the Fréchet distance, including its formal definition, weak and discrete variants, and algorithms for efficiently computing the distance. The report will also cover the general motivation for using the Fréchet distance and its applications in fields such as pattern recognition, computer vision, and geographic information systems. The report is structured to provide a comprehensive understanding of the Fréchet distance, including easy-to-understand explanations and a discussion of trade-offs between computational complexity and accuracy in the algorithms for computing the distance.

1 Introduction

At first, it seems that measuring the similarity between two curves is a very theoretical problem, but it has many, not that obvious, practical applications. It is used in geographic information systems (GIS) to quantify the similarity or quality of two maps [LWLL22], find patterns in vehicle traffic or animal tracking data by trajectory analysis [BBG+11], gesture recognition on touch devices like smartphones [HT22], and shape matching in the field of computer vision.

To quantify the similarity of two given curves (e.g. shapes, paths, or trajectories) a distance metric can be used. We can not simply use a distance metric like euclidean distance because we do not want to compare points to points but instead compare curves to curves. Therefore, we start by defining curves in Definition 1.

Definition 1 *We define a curve as a continuous function $f : [a, b] \mapsto V$ with $a, b \in \mathbb{R}, a \leq b$ and V being the space of the curve, e.g., $\mathbb{R}^2$ or $\mathbb{R}^3$.*

Definition 2 *We define a polygonal curve (polygonal chain) as a linearly interpolated curve on $n \in \mathbb{N}$ points in V. Therefore, $P : [0, n] \mapsto V$ should be affine on the interval $[i, i+1]$ for all $i \in \{0, 1, \ldots, n-1\}$, i.e., $P(i+\lambda) = (1-\lambda)P(i) + \lambda P(i+1)$ for all $\lambda \in [0, 1]$.*

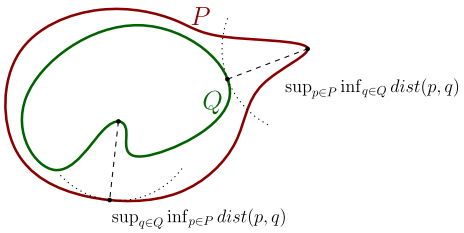
In the following we will focus on polygonal curves (polygonal chains) as defined in Definition 2 instead of classical curves (Definition 1), often used in pure mathematics, because most data we have in this area is captured by some noncontinuous mechanism, e.g., GPS positions from a tracking device. Additionally, we focus on the 2D Space, i.e. $V = \mathbb{R}^2$. However, many concepts also apply to normal curves and higher dimensional spaces.

The commonly known *Hausdorff distance* can be understood as the largest distance between any two points on the curves. For each curve, a point with the maximum shortest distance to a point on the other curve is searched for, see Figure 1a. [AG95]

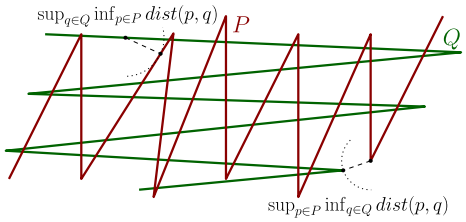
Definition 3 *Given two normal or polygonal curves P and Q, their Hausdorff distance $\delta_{hd}(P, Q)$ is defined as:*

$$\delta_{hd}(P, Q) = \max\left(\sup_{p \in P} \inf_{q \in Q} dist(p, q), \sup_{q \in Q} \inf_{p \in P} dist(p, q)\right)$$

The Hausdorff distance metric $\delta_{hd}(P, Q)$ defined in Definition 3 is very intuitive and can be easily computed for polygonal curves [Ko13]. It finds applications in computer vision and computer graphics where it is used to compare the similarity of two mesh objects [CRS98, Ko13]. However, this metric does not take the course of the curves into account, but only the point sets of the individual curves. Two curves may have a small Hausdorff distance but seem very dissimilar when looking at them, see Figure 1b.



(a) Hausdorff distance between the red curve P and green curve Q.



(b) Two dissimilar curves with a small Hausdorff distance.

Figure 1: Hausdorff distance examples.

In reality, the course of the curves is important for many applications. For example, in handwriting character recognition on digital input devices, especially in Chinese [Cha], (online) handwriting signature verification [ZGZH08, FW18], gesture recognition on smartphone touchscreens [HT22], map-matching tracking data for traffic control, routing and navigation [WSP06, BPSW05], moving object analysis, e.g., detecting commuting patterns [BBG+11], and motion evaluation in real-time motion capturing [QWL17, SD12]. For this reason, the *Fréchet distance*, named after the famous French mathematician René Maurice Fréchet, was introduced.

A common analogy for the Fréchet distance can be shown by a man walking his dog. Assume that both the man and the dog are moving along a predefined fixed curve. Both are free to adjust their speed as they wish, even to stop. However, they are not allowed to go backwards. We are looking for the minimum length of the dog leash that is necessary for both of them to reach their destination. The length of the dog leash is then exactly the Fréchet distance of the two curves. A formal definition is given in Definition 4 along with a visual example in Figure 2.

Definition 4 *Given two (polygonal) curves $P : [a, a'] \mapsto V$, $Q : [b, b'] \mapsto V$ and a distance function $dist : (V, V) \mapsto \mathbb{R}$, their Fréchet distance $\delta_F(P, Q)$ is defined as:*

$$\delta_F(P,Q) = \inf_{\substack{\alpha:[0,1]\mapsto[a,a'] \\ \beta:[0,1]\mapsto[b,b']}} \max_{t\in[0,1]} dist(P(\alpha(t)), Q(\beta(t)))$$

where α and β represent the walking speed as continuous monotonically increasing functions with $\alpha(0) = a, \alpha(1) = a'$, $\beta(0) = b, \beta(1) = b'$.

2 Variants

Through the years many slightly different variants of the continuous Fréchet distance (see Definition 4) have evolved. The continuous Fréchet distance takes into account every single point on the curves without allowing to walk backwards and without

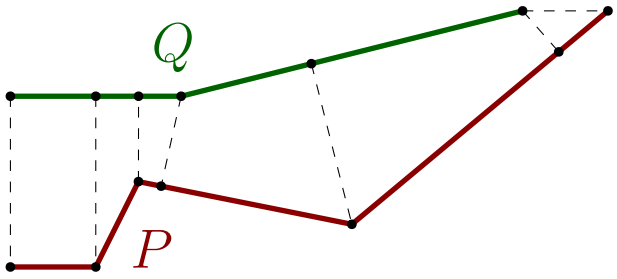


Figure 2: The Fréchet distance is indicated by the black lines connecting the two polygonal curves.

any obstacles in between the two curves. In this chapter the most prominent variants of the Fréchet distance are shortly described while algorithms for these are given in the following section 3.

2.1 Weak Fréchet Distance

The *weak Fréchet distance* (non-monotone Fréchet distance) slightly modifies the continuous version by allowing to go backwards. The definition remains the same as for the continuous Fréchet distance in Definition 4 where the monotonicity of α and β is no longer required. Note that they are still required to be continuous and have the correct starting and ending values. The weak Fréchet distance is a lower bound on the continuous Fréchet distance. However, the ratio between them may be arbitrarily large, as can be seen by the structure given in Figure 3. [AG95]

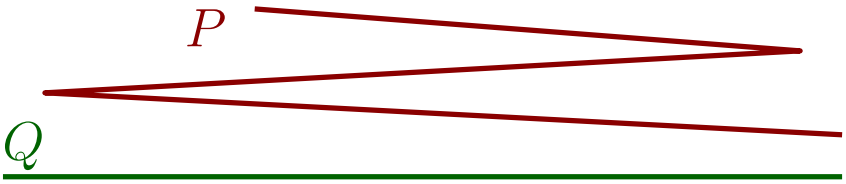


Figure 3: The two shown polygonal curves P and Q have a small weak Fréchet distance because the point on Q can be moved forward and backward as often as needed. The continuous Fréchet distance is rather large because the point on Q can only move forward.

2.2 Discrete Fréchet Distance

The *discrete Fréchet distance* is specifically tied to polygonal curves (Definition 2) because it requires a finite number of vertices and forbids walking or staying in between them. The dog walking analogy can be adapted by replacing both the man and the dog by two frogs, which are only allowed to jump from their current vertex to their next vertex in forward direction [Bri14]. While it is allowed for both frogs to jump simultaneously, it is not allowed to skip vertices while jumping. We ensure this behavior by defining a coupling between the curves in Definition 5.

Definition 5 *A coupling L between two polygonal curves P and Q, with points $(u_1, \ldots, u_p)$ and $(v_1, \ldots, v_q)$ respectively, is defined as a sequence*

$$(u_{a_1}, v_{b_1}), (u_{a_2}, v_{b_2}), \ldots, (u_{a_m}, v_{b_m})$$

such that $a_1 = 1, b_1 = 1, a_m = p, b_m = q$ *while preserving order and the jump length limitation. Thus, for all* $i \in \{1, \dots m-1\}$ *the jump length should be positive but no more than 1, i.e.,* $0 \leq a_{i+1} - a_i \leq 1$ *(respectively* $0 \leq b_{i+1} - b_i \leq 1$*).*

Note that P and Q may be different in terms of number of points, implying that the frog on the shorter curve has to stop and wait on at least one vertex once due to the pigeonhole principle. To order the possible couplings between P and Q we define the *length* of a coupling in Definition 6.

Definition 6 *The length* $||L||$ *of a coupling* L *is the greatest distance of the* m *pairs in* L*, that is,*

$$||L|| = \max_{i=1,\dots,m} dist(u_{a_i}, v_{b_i})$$

The definition of the discrete Fréchet distance immediately follows from the length of a coupling and is given by Definition 7.

Definition 7 *Given two polygonal curves* P *and* Q *their discrete Fréchet distance* $\delta_{dF}(P, Q)$ *is defined as the shortest coupling between them:*

$$\delta_{dF}(P, Q) = \min \{||L|| \mid L \text{ is a coupling between } P \text{ and } Q\}$$

The discrete variant (Definition 7) gives an upper bound on the continuous Fréchet distance (Definition 4). This might not be very obvious for the case of polygonal curves, but Figure 4 gives an intuitive example.

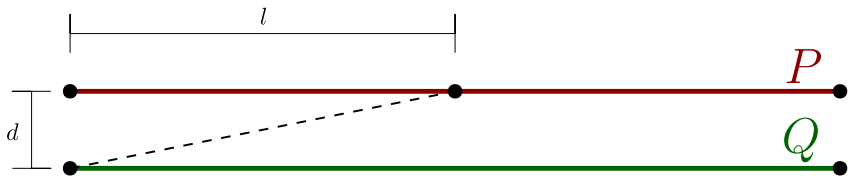


Figure 4: In the case of the discrete Fréchet distance one frog has to wait at either start or end of Q while the other one is at the middle vertex of P, thus requiring a minimal length of $\sqrt{d^2 + l^2}$ for any coupling. In contrast, a leash of length d is sufficient for the continuous Fréchet distance.

Additionally, it is also a lower bound on the continuous Fréchet distance plus the distance of the longest legal jump of any frog. By increasing the number of points of a polygonal curve that overapproximates some other polygonal curve this property can be used to compute the continuous Fréchet distance [EM94].

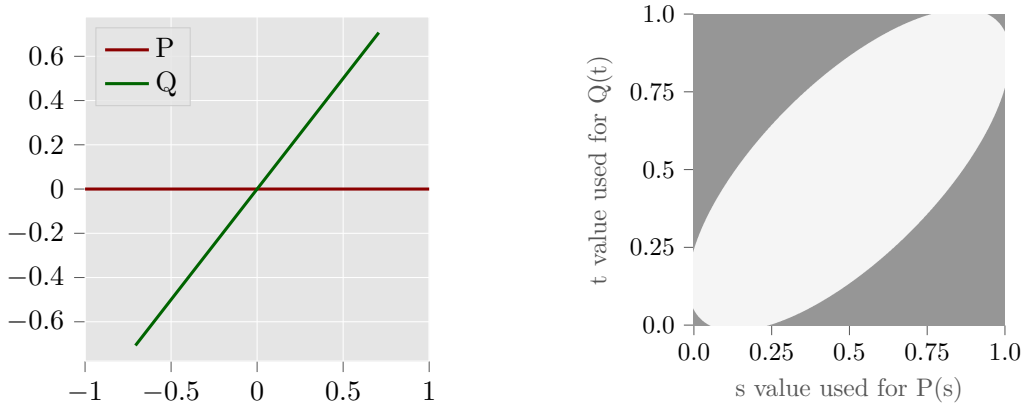
3 Algorithms

Several algorithms to compute the Fréchet distance of polygonal curves were proposed, starting from the original algorithm for the continuous variant by Alt and Godau in 1995 [AG95]. The algorithm is based on a concept called free-space diagram and gives a baseline for newer improved versions, e.g., the version by Buchin et al. [BBMM17]. This chapter is structured as follows: first the original algorithm for the continuous Fréchet distance is described in subsection 3.1. Afterwards, the modifications necessary to compute the weak variant are described in subsection 3.2. Finally, an algorithm to compute the discrete Fréchet distance is given in subsection 3.3.

3.1 Original Algorithm for the Continuous Fréchet Distance

The original algorithm to compute the continuous Fréchet distance (Definition 4) is based on the idea of solving a simpler decision problem and then doing a parameter search to find the exact value of δ_F. The decision problem is to decide whether two polygonal curves P and Q have a Fréchet distance $\delta_F(P, Q) \leq \varepsilon$ for some fixed $\varepsilon \geq 0$.

We start by looking at the simplest case for two polygonal curves with two vertices each, i.e., two line segments. The free space is the set of all valid combinations of positions on P and Q that have a distance of at most ε. Therefore, we define $F_\varepsilon = \{(s,t) \in [0,1]^2 \mid dist(P(s), Q(t)) \leq \varepsilon\}$. Plotting s and t on a two-dimensional graph, while indicating all illegal combinations in another color, gives the free space diagram, see Figure 5. [AG95]



(a) Line segments P and Q

(b) Free space diagram

Figure 5: The free space diagram Figure 5b for line segments P and Q and distance ε from Figure 5a using euclidean distance.

The free space F_ε has different shapes depending on the employed distance metric, e.g., the intersection of the unit square with an ellipse for euclidean distance and with a parallelogram for L_1 or L_∞. The exact structure of the free space is not further relevant, because only the convexity of the area is important. [AG95]

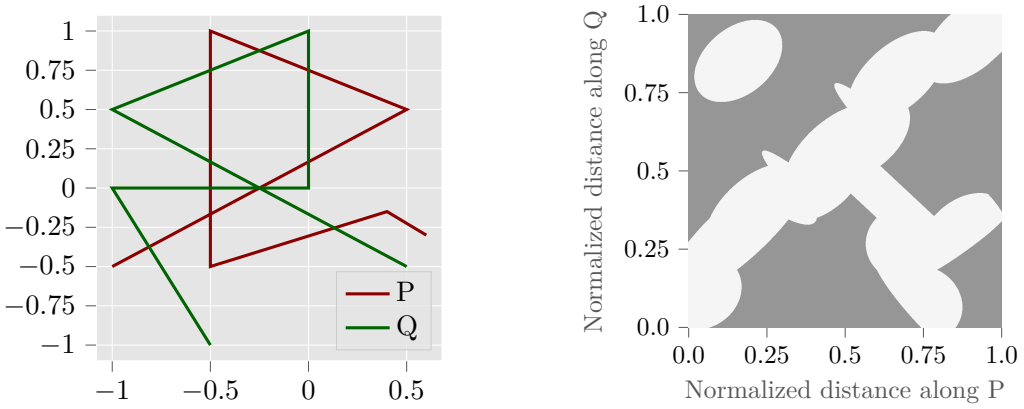
We can now extend the free space diagram to longer polygonal curves as stated in Definition 8. An example for a free space diagram on two non-trivial polygonal curves is given in Figure 6.

Definition 8 *Given two polygonal curves P and Q with $p, q \in \mathbb{N}$ vertices respectively, the free space is defined as:*

$$F_\varepsilon = \{(s,t) \in [0,p] \times [0,q] \mid dist(P(s), Q(t)) \leq \varepsilon\}$$

Computing the complete free space F_ε is very compute intensive because each point in the continuous space needs to be computed. Using the convexity property of each cell in the free space, we can observe that only a small constant number of points need to be computed per cell. These points define the legal regions on each boundary side of the cell, see Figure 7. We do not need to know the exact shape of the free space because we travel in a monotonically increasing way through a convex space, hence never entering any illegal regions.

Due to the constant number of intersection points required per cell, the free space of a single cell can be computed in $\mathcal{O}(1)$ time. Therefore, the complete free space of



(a) Polygonal curves P and Q

(b) Free space diagram

Figure 6: The free space diagram Figure 6b for polygonal curves P and Q from Figure 6a and distance $\varepsilon = 0.73$ using euclidean distance. (Figure inspired by [BBMM17])

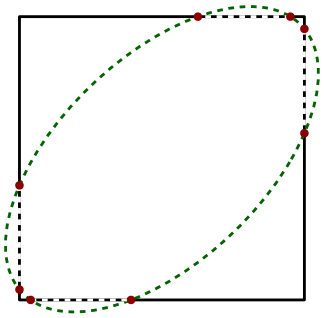


Figure 7: Legal intervals on the boundaries of a free space cell are shown with black white dashed lines. There are at most 8 intersection points for an ellipse and a rectangle. (Figure inspired by [AG95])

all cells can be computed in $\mathcal{O}(pq)$ time because there are $p-1$ (respectively $q-1$) edges in P (respectively Q).

We observe that there exists a monotonically increasing curve from $(0,0)$ to (p,q) in F_ε if and only if the Fréchet distance between P and Q is less than or equal to ε, i.e., $\delta_F(P,Q) \leq \varepsilon$. Given such a curve, the walking speed functions α and β from Definition 4 can be easily obtained by rescaling the parameter ranges. Computing such a monotonically increasing curve can be done using dynamic programming by computing the bottom left most reachable points in each cell starting at $(0,0)$. In each step the solution for a single cell can be computed in constant time based on the solutions to previously computed cells. Therefore, the runtime is also $\mathcal{O}(pq)$. See Figure 8 for an example curve. [AG95]

So far we discussed the decision problem whether $\delta_F(P,Q) \leq \varepsilon$ and observe that it is solvable in $\mathcal{O}(pq)$ time. What we actually want to know is the exact value of the Fréchet distance. A straight forward approach for this is to compute the value bit by bit using binary search in total $\mathcal{O}(pq \log(\text{“accuracy”})) = \mathcal{O}(pq(\text{“accuracy bits”}))$ time. In practice this approach works really well and is often favored over other more complicated approaches. [AG95]

Another approach is to check all values of ε where something “interesting” happens

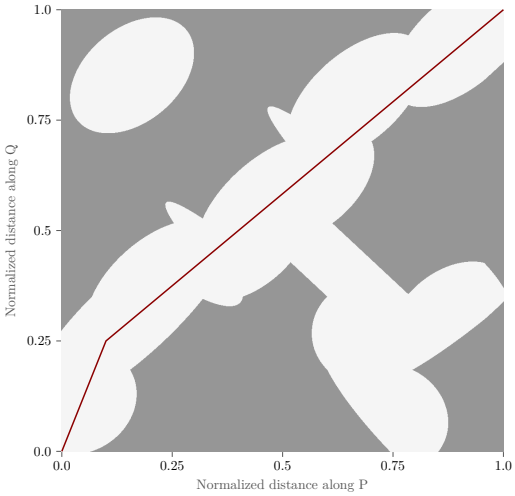


Figure 8: A monotonically increasing curve (red) through the free space diagram. (Figure inspired by [BBMM17])

until the smallest value of ε is obtained where a monotonically increasing path from $(0,0)$ to (p,q) in F_ε exists. We will call these the *critical values* of ε, which are:

1. the minimal value of ε where $(0,0) \in F_\varepsilon$ and $(p,q) \in F_\varepsilon$.
2. values of ε where a new passage on the border edge between two neighboring cells opens up.
3. values of ε where a new horizontal (respectively vertical) passage between two non-neighboring cells opens up.

There is one value of case 1, specifically the maximum of starting point to starting point and endpoint to endpoint distance, $\mathcal{O}(pq)$ values of case 2, namely the number of vertex to edge distances, and $\mathcal{O}(p^2q + pq^2)$ values of case 3, particularly the number of times two vertices of one curve have the same distance to one edge of the other curve. Determining all these values can be done in $\mathcal{O}(p^2q + pq^2)$ time because each value in each case can be computed in constant time, making the values of the third case dominate the time complexity. These values need to be sorted in $\mathcal{O}((p^2q + pq^2)\log(p^2q + pq^2)) = \mathcal{O}((p^2q + pq^2)\log(pq))$ time in order to perform binary search in $\mathcal{O}(pq\log(p^2q + pq^2))$ time on them using the algorithm for the decision problem. The overall time required for this approach is dominated by the sorting step, thus $\mathcal{O}((p^2q + pq^2)\log(pq))$. This time complexity can be decreased to $\mathcal{O}(pq\log(pq))$ using parametric search [Meg83, Col87] to work faster in theory. However, due to the specific internally used sorting network it is hard to implement this efficiently in practice. [AG95]

3.2 Algorithm for the Weak Fréchet Distance

For the weak Fréchet distance, as described in subsection 2.1, the computation becomes less complex compared to the continuous Fréchet distance. Specifically, we can ignore critical values of the third type, since it is no longer required to move monotonically and one could just go left or down in the free space diagram taking any arbitrary continuous path. Therefore, the focus shifts towards critical values of

the second type, those where new passages between neighboring cells open up. Alt and Godau [AG95] proposed two algorithms to compute the weak Fréchet distance.

For both algorithms the problem is modeled as a grid based graph of p by q vertices, one for each cell of the free space diagram, and edges connecting the vertices with their neighboring vertices. There may be up to four neighboring vertices, namely, top, bottom, left, and right. Additionally, s is added as the start vertex and t as the end vertex. This results in an undirected Graph $G = (V, E)$ where $V = \{C_{i,j} \mid 0 \leq i \leq p, 0 \leq j \leq q\} \cup \{s, t\}$ and $E = \{(C_{i,j}, C_{i+1,j}) \mid 0 \leq i < p, 0 \leq j \leq q\} \cup \{(C_{i,j}, C_{i,j+1}) \mid 0 \leq i \leq p, 0 \leq j < q\} \cup \{(s, C_{0,0}), (C_{p,q}, t)\}$. Each edge $(C_{i,j}, C_{i',j'})$ is assigned a critical minimal value of ε such that there is a passage through the border between two cells in the free space. Furthermore, the minimal value of ε such that $(0, 0) \in F_\varepsilon$ is assigned to the first edge $(s, C_{0,0})$ and the minimal value of ε such that $(p, q) \in F_\varepsilon$ is assigned to the last edge $(C_{p,q}, t)$. [AG95]

The first algorithm is again based on first solving the decision problem and then searching for the right ε through binary search. The decision problem is satisfied, if and only if there exists a path from s to t in G with a maximum single edge value of ε. This can be checked by removing all edges with values greater than ε from E and then checking if s is still connected to t. This can be done through some graph traversal algorithm like breadth-first search or depth-first search in $\mathcal{O}(pq)$ time. [AG95] Since the number of critical values is no longer dominated by the third case, rather by the second case of $\mathcal{O}(pq)$ values, the overall computation problem can be solved in $\mathcal{O}(pq \log(pq))$ time using binary search [vL13].

The second algorithm works by directly searching for a path from s to t in G, without any prior removal of edges from E. This can be done using Prim's minimum spanning tree algorithm [Pri57] starting at s and stopping as soon as t is included in the spanning tree. The value of ε is the value of the last added edge. Similarly, one can use Dijkstra's algorithm [Dij59], optimizing for the minimum maximum edge value. Both approaches result in a time complexity of $\mathcal{O}(pq \log(pq))$. [AG95] In this case of a planar grid based graph with non-negative edge values an even faster algorithm from Henzinger et al. [HKRS97] to solve the single source shortest paths can be applied. This gives a time complexity of $\mathcal{O}(pq)$ to compute the weak Fréchet distance [vL13].

3.3 Algorithm for the Discrete Fréchet Distance

The computation of the discrete Fréchet distance, as defined in Definition 7, is very easy and straight forward using a dynamic programming approach as originally suggested by Eiter et al. [EM94]. Recall that the polygonal curves $P = (u_1, \ldots, u_p)$ and $Q = (v_1, \ldots, v_p)$ are defined as sequences of points. The algorithm works by recursively computing the minimal required ε for partial curves $P' = (u_1, \ldots, u_i)$ and $Q' = (v_1, \ldots, v_j)$ using $c(i, j)$ as defined in Definition 9. Starting from $c(1, 1)$, which is the distance between the starting points u_1 and v_1, all other values of $c(i, j)$ are computed iteratively in a bottom up manner and memoization on the previously computed values of $c(i, j)$. The resulting algorithm has runtime of $\mathcal{O}(pq)$ because the constant time function $c(i, j)$ is called pq times [EM94].

Definition 9 *The discrete Fréchet distance of polygonal curves* $P' = (u_1, \ldots, u_i)$

and $Q' = (v_1, \ldots, v_j)$ *can be computed by*

$$c(i,j) = \begin{cases} dist(u_1, v_1), & \text{if } i = 1 \text{and } j = 1 \\ \max\{c(i-1,1), dist(u_i, v_1)\}, & \text{if } i > 1 \text{and } j = 1 \\ \max\{c(1,j-1), dist(u_1, v_j)\}, & \text{if } i = 1 \text{and } j > 1 \\ \max\{c(i-1,j), c(i,j-1), c(i-1,j-1), dist(u_i, v_j)\}, & \text{if } i > 1 \text{and } j > 1 \end{cases}$$
$$= \delta_{dF}(P', Q')$$

4 Conclusion

This work gives an overview on the Fréchet distance, its variants and algorithms to compute them. Several differences between the variants have been described and evaluated. In conclusion, it can be stated that computing the continuous Fréchet distance is more computationally expensive compared to its discrete and weak variants. However, computing the continuous variant can be done efficiently in practice by binary searching the bits of the desired accuracy directly instead of computing the formally correct value. Alternatively, the discrete Fréchet distance can be calculated, since it bounds the continuous distance from below and above. In this way, the tolerance range can be narrowed down to a desired size.

References

[AG95] Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *Int. J. Comput. Geom. Appl.*, 5:75–91, 1995.

[BBG+11] Kevin Buchin, Maike Buchin, Joachim Gudmundsson, Maarten Löffler, and Jun Luo. Detecting Commuting Patterns by Clustering Subtrajectories. *Int. J. Comput. Geom. Appl.*, 21(3):253–282, 2011.

[BBMM17] Kevin Buchin, Maike Buchin, Wouter Meulemans, and Wolfgang Mulzer. Four Soviets Walk the Dog-Improved Bounds for Computing the Fréchet Distance. *Discrete & Computational Geometry*, 58(1):180–216, July 2017.

[BPSW05] Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk. On Map-Matching Vehicle Tracking Data. In Klemens Böhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-Åke Larson, and Beng Chin Ooi, editors, *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, pages 853–864. ACM, 2005.

[Bri14] Karl Bringmann. Why Walking the Dog Takes Time: Frechet Distance Has No Strongly Subquadratic Algorithms Unless SETH Fails. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 661–670, Philadelphia, PA, USA, October 2014. IEEE.

[Cha] Chanin, David. Hanzi Writer. https://github.com/chanind/hanzi-writer, accessed December 2022.

[Col87] Richard Cole. Slowing down sorting networks to obtain faster sorting algorithms. *Journal of the ACM*, 34(1):200–208, January 1987.

[CRS98] P. Cignoni, C. Rocchini, and R. Scopigno. Metro: Measuring Error on Simplified Surfaces. *Computer Graphics Forum*, 17(2):167–174, 1998.

[Dij59] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[EM94] Thomas Eiter and Heikki Mannila. Computing Discrete Fréchet Distance. May 1994.

[FW18] J. L. Fang and W. Wu. Research on Signature Verification Method Based on Discrete Fréchet Distance. *IOP Conference Series: Materials Science and Engineering*, 359(1):012003, May 2018.

[HKRS97] Monika R Henzinger, Philip Klein, Satish Rao, and Sairam Subramanian. Faster Shortest-Path Algorithms for Planar Graphs. *Journal of Computer and System Sciences*, 55(1):3–23, August 1997.

[HT22] Shuxia Hu and Pei Tian. Research on Gesture Recognition Based on F-S Algorithm. In *2022 IEEE 6th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, pages 90–95, October 2022.

[Ko13] Ker-I Ko. On the complexity of computing the Hausdorff distance. *Journal of Complexity*, 29(3):248–262, June 2013.

[LWLL22] Cheng Lyu, Xinhua Wu, Yang Liu, and Zhiyuan Liu. A Partial-Fréchet-Distance-Based Framework for Bus Route Identification. *IEEE Transactions on Intelligent Transportation Systems*, 23(7):9275–9280, July 2022.

[Meg83] Nimrod Megiddo. Applying Parallel Computation Algorithms in the Design of Serial Algorithms. *Journal of the ACM*, 30(4):852–865, October 1983.

[Pri57] R. C. Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, November 1957. Conference Name: The Bell System Technical Journal.

[QWL17] Sen Qiao, Yilin Wang, and Jian Li. Real-time human gesture grading based on OpenPose. In *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pages 1–6, October 2017.

[SD12] Visar Shehu and Agni Dika. Curve similarity measurement algorithms for automatic gesture detection systems. In *2012 Proceedings of the 35th International Convention MIPRO*, pages 973–976, May 2012.

[vL13] Rolf van Leusden. A Novel Algorithm for Computing the Fréchet Distance. Master's thesis, Eindhoven University of Technology, May 2013.

[WSP06] C. Wenk, R. Salas, and D. Pfoser. Addressing the Need for Map-Matching Speed: Localizing Global Curve-Matching Algorithms. In *18th International Conference on Scientific and Statistical Database Management (SSDBM'06)*, pages 379–388, July 2006.

[ZGZH08] Jianbin Zheng, Xiaolei Gao, Enqi Zhan, and Zhangcan Huang. Algorithm of On-Line Handwriting Signature Verification Based on Discrete Fréchet Distance. In Lishan Kang, Zhihua Cai, Xuesong Yan, and Yong Liu, editors, *Advances in Computation and Intelligence*, pages 461–469, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.