Adaptive Computation of the Discrete Fréchet Distance

Jérémy Barbay(✉) iD

Departamento de Ciencias de la Computación, University of Chile, Santiago, Chile
jeremy@barbay.cl

Abstract. The discrete Fréchet distance is a measure of similarity between point sequences which permits to abstract differences of resolution between the two curves, approximating the original Fréchet distance between curves. Such distance between sequences of respective length n and m can be computed in time within $O(nm)$ and space within $O(n + m)$ using classical dynamic programming techniques, a complexity likely to be optimal in the worst case over sequences of similar length unless the Strong Exponential Time Hypothesis is proved incorrect. We propose a parameterized analysis of the computational complexity of the discrete Fréchet distance in function of the area of the dynamic program matrix relevant to the computation, measured by its *certificate width* ω. We prove that the discrete Fréchet distance can be computed in time within $O((n + m)\omega)$ and space within $O(n + m)$.

Keywords: Adaptive algorithm · Dynamic programming
Fréchet distance

1 Introduction

Measuring the similarity between two curves has applications in areas such as handwriting recognition [14], protein structure alignment [10], quantifying macro-molecular pathways [13], morphing [3], movement analysis [7], and many others [15]. One of the most popular solutions, the FRÉCHET DISTANCE is a measure of similarity between two curves $\mathcal{P}$ and $\mathcal{Q}$, that takes into account the location and ordering of the points along the curves. It permits, among other features, to abstract the difference of resolution between $\mathcal{P}$ and $\mathcal{Q}$, with application to morphing, handwriting recognition and protein structure alignment, among others [15]. In 1995, Art and Godau [9] described an algorithm computing the FRÉCHET DISTANCE between two polygonal curves composed of n and m segments respectively in time within $O(mn \log(mn))$.

One year before (1994), Eiter and Mannila [4] had extended the notion of the FRÉCHET DISTANCE between curves to the DISCRETE FRÉCHET DISTANCE

---

between sequences of points of respective sizes n and m, demonstrating that the latter can be used to approximate the former in time within $O(nm)$ and space within $O(n + m)$ using classical dynamic programming techniques. Two decades later (2014), Bringmann [2] showed that this worst case complexity is likely to be optimal, unless a bunch of other problems (among which CNF SAT) can be computed faster than usually expected. Hence, the bounds about the computational complexity of the DISCRETE FRÉCHET DISTANCE in the worst case over instances of input sizes n and m are reasonably tight.

Yet, for various restricted classes of curves (e.g. κ-bounded, backbone, c-packed and long-edged [8] curves), both the FRÉCHET DISTANCE and the DISCRETE FRÉCHET DISTANCE are known to be easier to compute. Among other examples, we consider the FRÉCHET DISTANCE DECISION problem, which consists in deciding whether the FRÉCHET DISTANCE between two curves is equal to a given value f. In 2018, Gudmundsson et al. [8] described an algorithm deciding if the FRÉCHET DISTANCE is equal to a given value f in time linear in the size of the input curves when each edge is longer than the FRÉCHET DISTANCE between those two curves. Those results easily project to the DISCRETE FRÉCHET DISTANCE.

Those results for the mere computation of the DISCRETE FRÉCHET DISTANCE suggest that one does not always need to compute the $n \times m$ values of the dynamic program. **Can such approaches be applied to more general instances**, such that the **area of the dynamic program which *needs* to be computed** measures **the *difficulty* of the instance**?

In this work we perform a parameterized analysis of the computational complexity of the DISCRETE FRÉCHET DISTANCE, in function of the area of the dynamic program matrix relevant to the computation, measured by its *certificate width* ω. After describing summarily the traditional way to compute the DISCRETE FRÉCHET DISTANCE and the particular case of long edged curves (Sect. 2), we describe an optimization of the classical dynamic program based on two simple techniques, banded dynamic programming and thresholding (Sect. 3), and we prove that this program runs in time within $O((n + m)\omega)$ and space within $O(n + m)$ (Sect. 4). We conclude with a discussion in Sect. 5 of how our results generalize those of Gudmundsson et al. [8], and the potential applications and generalizations of our techniques to other problems where dynamic programs have given good results.

2   Preliminaries

Before describing our results, we describe some classical results upon which we build: the classical dynamic program computing the DISCRETE FRÉCHET DISTANCE, and the "easy" case of long-edged curves described by Gudmundsson et al. [8].

Classical Dynamic Program: Let $P[1..n]$ and $Q[1..m]$ be sequences of n and m points with $n \geq m$. The DISCRETE FRÉCHET DISTANCE between P and Q is

the minimal width of a traversal of P and Q, where the width of a traversal is the maximal distance separating two points $u \in P$ and $v \in Q$, where u and v progress independently, but always forward.

Such a distance is easily computed using classical techniques from dynamic programming. Algorithm 1 (page 4) describes a simple implementation in `Python`, executing in time within $O(nm)$.

Algorithm 1 Classical algorithm to compute the DISCRETE FRÉCHET DISTANCE between two sequences of points P and Q. The implementation is decomposed in two parts: the `computation` function initializes the array of values, which is filled recursively by the `recursive` function. For the sake of space, the documentation strings and unit tests were not included, but the source file including those is available at https://gitlab.com/FineGrainedAnalysis/Frechet.

```python
def recursive(dpA,i,j,P,Q):
    if dpA[i,j] > -1:
        return dpA[i,j]
    elif i == 0 and j == 0:
        dpA[i,j] = distance(P[0],Q[0])
    elif i > 0 and j == 0:
        dpA[i,j] = max(
            recursive(dpA,i-1,0,P,Q),
            distance(P[i],Q[0]))
    elif i == 0 and j > 0:
        dpA[i,j] = max(
            recursive(dpA,0,j-1,P,Q),
            distance(P[0],Q[j]))
    elif i > 0 and j > 0:
        dpA[i,j] = max(
            min(
                recursive(dpA,i-1,j,P,Q),
                recursive(dpA,i-1,j-1,P,Q),
                recursive(dpA,i,j-1,P,Q)),
            distance(P[i],Q[j]))
    else:
        dpA[i,j] = float("inf")
    return dpA[i,j]

def computation(P,Q):
    dpA = numpy.ones((len(P),len(Q)))
    dpA = numpy.multiply(dpA,-1)
    d = recursive(dpA,len(P)-1,len(Q)-1,P,Q)
    return d
```

While such a simple algorithm also requires space within $O(nm)$, a simple optimization yields a space within $O(n + m)$, by computing the DISCRETE FRÉCHET DISTANCE between $P[1..i]$ and $Q[1..j]$ for increasing values of i and j,

one column and row at the time, keeping in memory only the previous column and row. We describe in Sect. 3 a more sophisticated algorithm which avoids computing some of the $n \times m$ values computed by Algorithm 1.

Easy Instances of the Fréchet Distance: For various restricted classes of curves, such as long-edged [8] curves, both the FRÉCHET DISTANCE and the DISCRETE FRÉCHET DISTANCE are known to be easier to compute (or approximate). In 2018, Gudmundsson et al. [8] showed that in the special case where all the edges of the polygonal curve are longer than the FRÉCHET DISTANCE, the latter can be decided (i.e., checking a value of the FRÉCHET DISTANCE) in linear time in the size of the input, computed in time within $O((n + m) \lg(n + m))$.

In the next section, we describe a quite simple algorithm which not only takes advantage of long edged curves, but of any pair of curves for which a consequent part of the array of the dynamic program can be ignored.

3   An Opportunistic Dynamic Program

We describe an algorithm based on two complementary techniques: first, a *banded dynamic program*, which *approximates* the value computed by a classical dynamic program by considering only the values of the dynamic program within a range of width w (for some parameter w) around the diagonal (a technique previously introduced for the computation of the EDIT DISTANCE between two strings); and second, a *thresholding* process, which *accelerates* the computation by cutting the recurrence any time the distance computed becomes larger or equal to a threshold t (for some parameter t corresponding to a distance already achieved by some traversal of the two curves). The combination of those two techniques, combined with a parametrization of the problem, yields the parameterized upper bound on the computational complexity of the DISCRETE FRÉCHET DISTANCE.

Banded Dynamic Program: When computing the EDIT DISTANCE (e.g., the DELETE INSERT EDIT DISTANCE, or the LEVENSHTEIN EDIT DISTANCE [1]) between similar strings $S \in [1..\sigma]^n$ and $T \in [1..\sigma]^m$ (i.e., their EDIT DISTANCE d is small), it is possible to compute less than $n \times m$ cells of the dynamic program array, and hence compute the EDIT DISTANCE in time within $O(d(n + m)) \subseteq O(nm)$. The "trick" is based on the following observation: when the distance between the two strings is d, the "paths" corresponding to d operations transforming S into T in the matrix of the dynamic program errs at most at distance d from the diagonal between the cell $(1, 1)$ and the cell (n, m). Based on this observation, it is sufficient to compute the number of operations corresponding to paths inside a "band" of width d around such a diagonal [1]. This technique needs some adaptation to be applied to the computation of the DISCRETE FRÉCHET DISTANCE f between two curves, for two reasons: first, f is a real number (it corresponds to the Euclidean distance between two points) and not an integer as the number of edition operations, and this number is independent

of the number of cells of the dynamic program being computed; and second, the definition of the DISCRETE FRÉCHET DISTANCE is based on a maximum rather than a sum, which actually makes another optimization possible, described in the next paragraph.

Thresholding: Given two sequences of points $P[1..n]$ and $Q[1..m]$, consider the *Euclidean matrix* $E(P, Q)$ of all $n \times m$ distances between a point $u \in P$ and a point $v \in Q$. Any parallel traversal of P and Q corresponds to a path in $E(P, Q)$ from the top left cell $(1, 1)$ to the bottom right cell (n, m); the width of such a traversal is the maximum value in $E(P, Q)$ on this path; and the DISCRETE FRÉCHET DISTANCE is the minimum width achieved over all such traversals.

Now suppose that, as for the EDIT DISTANCE between two similar strings, the traversal of the Euclidean matrix $E(P, Q)$ corresponding to the DISCRETE FRÉCHET DISTANCE f between P and Q is close to the diagonal from $(1, 1)$ to (n, m), and that any traversal diverging from such a path "encounters" a pair of points (u, v) at euclidean distance larger than f (in particular, this happens when the two curves are "long edged" compared to f). Then, some of the values of the cells of the dynamic program matrix outside of this diagonal can be ignored for the computation of the DISCRETE FRÉCHET DISTANCE between P and Q.

In the following paragraph we describe how to combine those two techniques into an adaptive algorithm taking advantage of "easy" instances where a large quantity of cells of the dynamic program can be ignored.

Combining the Two Techniques: The solution described consists of two algorithms: an approximation Algorithm 2 which computes a parameterized upper bound on the DISCRETE FRÉCHET DISTANCE, and a computation Algorithm 3 which calls the previous one iteratively with various parameter values, in order to compute the real DISCRETE FRÉCHET DISTANCE of the instance.

Algorithm 2 lists an implementation in `Python` of an algorithm which, given as parameters two arrays of points P and Q, an `integer` *width w*, and a `float` *threshold t*; computes an upper bound of the DISCRETE FRÉCHET DISTANCE between P and Q, obtained by computing only the cells within a band of width $2w$ around the diagonal from the top left cell $(1, 1)$ to the bottom right cell (n, m), and cutting all sequences of recursive calls when reaching a distance of value t or above. This algorithm uses space within $(n + m)$ as it computes the values from $(1, 1)$ to (n, m) by updating and switching alternatively two arrays of size n and two arrays of size m (respectively corresponding to rows and columns of the dynamic program matrix). Its running time is within $O(w(n + m))$, as it computes at most $w(n + m)$ cells of the dynamic program array. Furthermore, it not only returns the value of the upper bound computed, but also a Boolean `breached` indicating if the border of the banded diagonal has been reached during this computation. When such a border has *not* been reached (and the threshold value t is smaller than or equal to the DISCRETE FRÉCHET DISTANCE between P and Q), the value returned is equal to the DISCRETE FRÉCHET DISTANCE between P and Q.

Algorithm 2 Parameterized Algorithm to approximate the DISCRETE FRÉCHET DISTANCE between two sequences of points by computing only values of the dynamic program within a band of width w around the diagonal, and limiting the recursion to distances smaller than a threshold t.

```python
def approximation(P,Q,w,t):
    bReached = False
    n = len(P)
    m = len(Q)
    assert( m <= n )
    assert( m > 0 )
    def e(i,j):
        d = utils.distance(P[i],Q[j])
        if d < t:
            if (i-j) >= w or (j-i) >= w:
                bReached = True
            return d
        else:
            return float("inf")
    oldRow = np.ones(n)
    oldColumn = np.ones(m)
    oldRow[0] = oldColumn[0] = e(0,0)
    for s in range(1,m):
        newRow = np.ones(n)
        for i in range(max(1,s-w+1),s):
            newRow[i]  = max(e(i,s),min(
                oldRow[i],
                oldRow[i-1],
                newRow[i-1]))
        newColumn = np.ones(m)
        for j in range(max(1,s-w+1),s):
            newColumn[j]  = max(e(s,j),min(
                newColumn[j-1],
                oldColumn[j-1],
                oldColumn[j]))
        newColumn[s] = newRow[s] = max(e(s,s),min(
            newRow[s-1],
            newColumn[s-1],
            oldRow[s-1]))
        oldRow = newRow
        oldColumn = newColumn
    for s in range(m,n):
        newColumn = np.ones(m)
        for j in range(max(1,s-w+1),m):
            newColumn[j]  = max(e(s,j),min(
                oldColumn[j],
                oldColumn[j-1],
                newColumn[j-1]))
        oldColumn = newColumn
    return bReached,newRow[n-1]
```

Algorithm 3 lists an implementation in `Python` of an algorithm which, given as parameters two sequences of points P and Q, calls the `approximation` Algorithm 2 on P and Q for widths of exponentially increasing value (by a factor of two). The first call is performed with an infinite threshold (no information is available on the similarity of the curve at this point), but each successive call uses the best upper bound on the DISCRETE FRÉCHET DISTANCE between P and Q previously computed as a threshold.

Algorithm 3 Adaptive algorithm to compute the DISCRETE FRÉCHET DISTANCE between two sequences of points, by iteratively approximating it with increasing width, using the value of the previous approximation to potentially reduce the number of distances being computed.

```python
def computation(P,Q):
    if len(P)<len(Q):
        P,Q = Q,P
    if len(Q) == 0:
        return float("inf")
    w = 1
    bReached,t=approximation(P,Q,w,float("inf"))
    while bReached and w < len(Q):
        w = 2*x
        bReached,t=approximation(P,Q,w,t)
    return t
```

In the next section, we analyze the running time of Algorithm 3, and describe a parameterized upper bound on it.

4   Parameterized Upper Bound

The running time of the approximation Algorithm 2 when given parameter w is clearly within $O(w(n + m))$: it computes within $O(w)$ cells in at most $n + m$ rounds, each in constant time. A finer upper bound taking into account the value of the parameter t requires more hypothesis on the relation between P and Q, for which we need to consider the running time of the `computation` Algorithm 3. We model such hypothesis on the instance in the form of a *certificate*, and more specifically in the form of a *certificate area* of the Euclidean matrix corresponding to a set of values which suffice to *certify* the value of the DISCRETE FRÉCHET DISTANCE.

Definition 1. *Given two sequences of points $P[1..n]$ and $Q[1..m]$ of respective lengths n and m and of DISCRETE FRÉCHET DISTANCE f, a* Certificate Area *of the instance formed by P and Q is an area of the Euclidean matrix of P and Q containing both $(1, 1)$ and (n, m), and delimited by two paths (one above and one below), both such that the minimum value on this path is larger than or equal to f. The* width *of such a certificate area is the minimal* width *of a banded diagonal containing both paths.*

The surface of such an area is a good measure of the difficulty to certify the DISCRETE FRÉCHET DISTANCE, but the minimal width of such an area lends itself better to an analysis of the running time of the computation Algorithm 3:

Definition 2. *Given two sequences of points $P[1..n]$ and $Q[1..m]$, the* Certificate Width ω *of (P,Q) is the minimum width of a certificate area, taken over all possible certificate areas of (P,Q).*

Such a width can be as large as $n + m$ in the worst case over instances formed by sequences of points of respective lengths n and m, but the smaller this *certificate width* is, the faster Algorithm 3 runs:

Theorem 1. *Given two sequences of points $P[1..n]$ and $Q[1..m]$ forming an instance of certificate width ω, Algorithm 3 computes the* DISCRETE FRÉCHET DISTANCE *between P and Q in time within $O((n + m)\omega)$ and space within $O(n + m)$.*

Beyond the necessity to measure experimentally the certificate width of practical instances of the DISCRETE FRÉCHET DISTANCE, and the exact running time of Algorithm 3 on such instances, we discuss some more subtle options for future work in the next section.

5   Discussion

The results described in this work are by far only preliminary. Among the various questions that those preliminary results raise, we discuss here the relation to the long edged sequences recently described by Gudmundsson et al. [8]; a potential parameterized conditional lower bound matching our parameterized upper bound on the computational complexity of the DISCRETE FRÉCHET DISTANCE; (the not so) similar results on the ORTHOGONAL VECTOR decision problem; and the possibility of a theory of reductions between parameterized versions of polynomial problems without clear (parameterized or not) computational complexity lower bounds.

Relation to Long Edged Sequences: In 2018, Gudmundsson et al. [8] described an algorithm deciding if the Fréchet distance is equal to a given value f in time linear in the size of the input curves when each edge is longer than the FRÉCHET DISTANCE between those two curves. Algorithm 3 is more general than Gudmundsson et al.'s algorithm [8], but it also performs in linear time on long-edged instances: the traversal corresponding to the FRÉCHET DISTANCE of such an instance is along the diagonal, implying a *certificate width* of 1. See Figs. 1, 2 and 3 for the Euclidean matrix, Fréchet Matrix and Dynamic Program Matrix of a random instance formed of 5 points, each edge of length 100 with a FRÉCHET DISTANCE of 13.45.

The ratio between the FRÉCHET DISTANCE and the minimal edge length of the curves might prove to be a more "natural" parameter than the *certificate width* to measure the "difficulty" of computing the FRÉCHET DISTANCE of a pair of curves: we focused on the *certificate width* in the hope that such a technique can find applications in the analysis of other problems on which dynamic programming has yield good solutions (Fig. 4).

1.41	101.43	193.97	294.66	199.17	227.36
100.14	**13.45**	94.67	195.37	99.5	142.89
199.48	97.99	**5.39**	105.43	6.13	113.25
290.44	192.56	104.04	**6.0**	97.26	109.6
193.23	93.14	13.17	104.98	**10.44**	99.05
232.69	156.27	112.89	104.58	107.64	**6.4**

Fig. 1. Euclidean matrix for a long edged instance: the 6 points from the first curve were randomly generated at distance 100 of each other, while the points from the second curve were generated by perturbing within a distance of 10 from the points of the first curve.

1.41	101.43	193.97	294.66	294.66	294.66
100.14	**13.45**	94.67	195.37	195.37	195.37
199.48	97.99	**13.45**	105.43	105.43	113.25
290.44	192.56	104.04	**13.45**	97.26	109.6
290.44	192.56	104.04	104.98	**13.45**	99.05
290.44	192.56	112.89	104.58	107.64	**13.45**

Fig. 2. Fréchet matrix for the same long edged instance as Fig. 1: the traversal corresponding to the FRÉCHET DISTANCE of the instance is along the diagonal (highlighted in bold here), resulting in a FRÉCHET DISTANCE of 13.45.

1.41	inf	inf	-1.0	-1.0	-1.0
inf	**13.45**	inf	inf	-1.0	-1.0
inf	inf	**13.45**	inf	6.13	-1.0
-1.0	inf	inf	**13.45**	inf	inf
-1.0	-1.0	13.17	inf	**13.45**	inf
-1.0	-1.0	-1.0	inf	inf	**13.45**

Fig. 3. Dynamic program matrix for the same long edged instance as Fig. 1, with width 3 and threshold 20: "inf" denotes interrupted recurrences because the distance found is already larger than the threshold, meanwhile values outside of the band of width 3 are marked with "−1".

9.43	19.48	19.48	-1.0	-1.0	-1.0
18.81	**11.31**	11.31	16.86	-1.0	-1.0
18.81	14.26	**11.31**	11.31	inf	-1.0
-1.0	inf	16.07	**11.31**	16.5	13.77
-1.0	-1.0	inf	11.31	**11.31**	11.31
-1.0	-1.0	-1.0	3.17	14.06	**11.31**

Fig. 4. Dynamic program matrix for a general instance. The 6 points from the first curve were randomly generated at distance 10 of each other, the points from the second curve by perturbing within a distance of 10 the points of the first curve. The computation of the matrix is performed with width 3 and threshold 20 as before.

Parameterized Conditional Lower Bound: The original motivation for this work was to prove a parameterized conditional lower bound on the computational complexity of the DISCRETE FRÉCHET DISTANCE as a step-stone for doing the same for the computation of various EDIT DISTANCES. The first step in this direction was the identification of a parameter for this problem: the *certificate width*, that seems to be a good candidate. The next step is to refine the reduction from

CNF SAT to the DISCRETE FRÉCHET DISTANCE described by Bringmann [2], in order to define a reduction from (a potential parameterized version of) CNF SAT to a parameterized version of the DISCRETE FRÉCHET DISTANCE.

Parameterized Upper and Lower Bound for the Computation of Orthogonal Vectors: Bringmann [2] mentions that the reduction from SAT CNF to the computation of the DISCRETE FRÉCHET DISTANCE is similar to Williams' reduction from SAT CNF to the (polynomial) problem of deciding if two sets of vectors contain an ORTHOGONAL VECTOR pair, and that there might be a reduction from the ORTHOGONAL VECTOR decision problem to the computation of the DISCRETE FRÉCHET DISTANCE. This mention called the ORTHOGONAL VECTOR decision problem to our attention, and in particular (1) the possibility of a parametrization of the analysis of this problem, and (2) a potential linear (or parameterized) reduction from such a parameterized ORTHOGONAL VECTOR decision problem to the parameterized computation of the DISCRETE FRÉCHET DISTANCE described in this work. It turns out that there exists an algorithm solving the ORTHOGONAL VECTOR decision problem in time within $O((n + m)(\delta + \log(n) + \log(m)))$, where n and m are the respective sizes of the sets of vectors forming the instance, and δ is the *certificate density* measuring the proportion of cells from the dynamic program which are sufficient to compute in order to certify the answer to the program. The reduction of this to the DISCRETE FRÉCHET DISTANCE will be more problematic: the two measures of difficulty seem completely unrelated.

A Theory of Reduction Between Polynomial Parameterized Problems: The study of the *parameterized complexity* of NP-hard problems [6,11] yields a theory of reduction between pairs formed by a decision problem P and a parameter k. The study of *adaptive sorting* algorithms [5,12] yields a theory of reductions between parameters measuring the existing disorder in an array to be sorted (which can also be seen as a theory of reductions between pairs of problems and parameters, but where all the problems are equal). Considering the theory of reductions between polynomial problems such as the DISCRETE FRÉCHET DISTANCE, the various EDIT DISTANCES between strings, the ORTHOGONAL VECTOR decision problem, and many others, one can imagine that it would be possible to define a theory of reductions between parameterized versions of these problems.

Data and Material Availability. The sources of this article, along with the unabridged code, will be made publicly available upon publication at the url https://gitlab.com/FineGrainedAnalysis/Frechet.

References

1. Bergroth, L., Hakonen, H., Raita, T.: A survey of longest common subsequence algorithms. In: Proceedings of the 11th Symposium on String Processing and Information Retrieval (SPIRE), pp. 39–48 (2000)
2. Bringmann, K.: Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In: Proceedings of the 2014 IEEE 55th Annual Symposium on Foundations of Computer Science, FOCS 2014, pp. 661–670. IEEE Computer Society, Washington, DC (2014)
3. Efrat, A., Guibas, L., Har-Peled, S., Mitchell, J., Murali, T.: New similarity measures between polylines with applications to morphing and polygon sweeping. Discret. Comput. Geom. **28**(4), 535–569 (2002)
4. Eiter, T., Mannila, H.: Computing discrete Fréchet distance. Technical report, Christian Doppler Labor für Expertensyteme, Technische Universität Wien (1994)
5. Estivill-Castro, V., Wood, D.: A survey of adaptive sorting algorithms. ACM Comput. Surv. (ACMCS) **24**(4), 441–476 (1992)
6. Flum, J., Grohe, M.: Parameterized Complexity Theory. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag New York Inc., Secaucus (2006). https://doi.org/10.1007/3-540-29953-X
7. Gudmundsson, J., Laube, P., Wolle, T.: Movement patterns in spatio-temporal data. In: Shekhar, S., Xiong, H. (eds.) Encyclopedia of GIS, pp. 1362–1370. Springer, Cham (2017). https://doi.org/10.1007/978-0-387-35973-1_823
8. Gudmundsson, J., Mirzanezhad, M., Mohades, A., Wenk, C.: Fast Fréchet distance between curves with long edges. In: Proceedings of the International Workshop on Interactive and Spatial Computing (WISC), IWISC 2018, pp. 52–58. ACM, New York (2018). Best Paper Award IWISC'18
9. Alt, H., Godau, M.: Computing the Fréchet distance between two polygonal curves. Int. J. Comput. Geom. Appl. (IJCGA) **5**(1–2), 75–91 (1995)
10. Jiang, M., Xu, Y., Zhu, B.: Protein structure-structure alignment with discrete Fréchet distance. J. Bioinform. Comput. Biol. **6**(1), 51–64 (2008)
11. Marx, D.: Parameterized complexity of constraint satisfaction problems. In: Proceedings of 19th Annual IEEE Conference on Computational Complexity (CCC), pp. 139–149 (2004)
12. Moffat, A., Petersson, O.: An overview of adaptive sorting. Aust. Comput. J. (ACJ) **24**(2), 70–77 (1992)
13. Seyler, S.L., Kumar, A., Thorpe, M.F., Beckstein, O.: Path similarity analysis: a method for quantifying macromolecular pathways. PLoS Comput. Biol. **11**, e1004568 (2015)
14. Sriraghavendra, E., Karthik, K., Bhattacharyya, C.: Fréchet distance based approach for searching online handwritten documents. In: Proceedings of the Ninth International Conference on Document Analysis and Recognition, ICDAR 2007, vol. 01, pp. 461–465. IEEE Computer Society, Washington, DC (2007). http://dl.acm.org/citation.cfm?id=1304595.1304769
15. Wikipedia: Frechet_distance. https://en.wikipedia.org/wiki/Fr'echet_distance