Algorithms for the Discrete Fréchet Distance Under Translation

Omrit Filtser[1]
Department of Computer Science, Ben-Gurion University of the Negev
Beer-Sheva 84105, Israel
omritna@cs.bgu.ac.il

Matthew J. Katz[2]
Department of Computer Science, Ben-Gurion University of the Negev
Beer-Sheva 84105, Israel
matya@cs.bgu.ac.il

Abstract

The (discrete) Fréchet distance (DFD) is a popular similarity measure for curves. Often the input curves are not aligned, so one of them must undergo some transformation for the distance computation to be meaningful. Ben Avraham et al. [5] presented an $O(m^3n^2(1+\log(n/m))\log(m+n))$-time algorithm for DFD between two sequences of points of sizes m and n in the plane under translation. In this paper we consider two variants of DFD, both under translation.

For DFD with shortcuts in the plane, we present an $O(m^2n^2\log^2(m+n))$-time algorithm, by presenting a dynamic data structure for reachability queries in the underlying directed graph. In 1D, we show how to avoid the use of parametric search and remove a logarithmic factor from the running time of (the 1D versions of) these algorithms and of an algorithm for the weak discrete Fréchet distance; the resulting running times are thus $O(m^2n(1+\log(n/m)))$, for the discrete Fréchet distance, and $O(mn\log(m+n))$, for its two variants.

Our 1D algorithms follow a general scheme introduced by Martello et al. [21] for the Balanced Optimization Problem (BOP), which is especially useful when an efficient dynamic version of the feasibility decider is available. We present an alternative scheme for BOP, whose advantage is that it yields efficient algorithms quite easily, without having to devise a specially tailored dynamic version of the feasibility decider. We demonstrate our scheme on the *most uniform path* problem (significantly improving the known bound), and observe that the weak DFD under translation in 1D is a special case of it.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases curve similarity, discrete Fréchet distance, translation, algorithms, BOP

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.20

1 Introduction

Polygonal curves play an important role in many applied domains, and it is a challenging task to compare them in a way that will reflect our intuitive notion of resemblance. The *Fréchet distance* is a useful and well studied similarity measure that has been applied in many diverse settings. Consider a man and a dog connected by a leash, each walking along a curve. They can control their speed but they are not allowed to backtrack. The Fréchet

[1] O. Filtser was supported by the Ministry of Science, Technology & Space, Israel, and by the Lynn and William Frankel Center.
[2] M. Katz was supported by grant 1884/16 from the Israel Science Foundation and grant 2014/170 from the US-Israel Binational Science Foundation.

distance between the two curves is the minimum length of a leash that is sufficient for such a dog-walk from the starting points to the end points of the curves.

Intuitively, the *discrete Fréchet distance* (DFD) replaces the curves by two sequences of points $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_m)$, and replaces the man and dog by two frogs (connected by a leash), the A-frog and the B-frog, initially placed at a_1 and b_1, respectively. At each move, the A-frog or the B-frog (or both) jumps from its current point to the next one. We are interested in the minimum length of a leash that allows the A-frog and the B-frog to reach a_n and b_m, respectively. The discrete distance is considered a good approximation of the continuous one, and is somewhat easier to compute. Both versions of the Fréchet distance (continuous and discrete) can be computed in roughly $O(n^2)$-time [1, 2, 7, 8, 14].

When the curves or the sampled sequences of points are generated by physical sensors, such as GPS devices, inaccurate measurements may occur. Since the Fréchet distance is a bottleneck measure and is thus very sensitive to outliers, several variants for handling outliers have been proposed, among these are: average and summed Fréchet distance [6, 10, 13], partial Fréchet similarity [9], and Fréchet distance with shortcuts [4, 11, 12].

In the *(one-sided) discrete Fréchet distance with shortcuts* (DFDS), we allow the A-frog to jump to any point that comes later in its sequence, rather than to only the next point. The B-frog has to visit all the B points in order, as in the standard discrete Fréchet distance. Driemel and Har-Peled [12] introduced the (continuous) Fréchet distance with shortcuts. They considered the vertex-restricted version where the dog is allowed to take shortcuts only by walking from a vertex v to any succeeding vertex w along the line segment connecting v and w, and presented an $O(n^5 \log n)$-time algorithm for this version. Later, Buchin et al. [11] showed that in the general case, where the dog is allowed to take shortcuts between any two points on its (continuous) curve, the problem becomes NP-hard. In the discrete case, however, the situation is much better. Ben Avraham et al. [4] presented an $O((m+n)^{6/5+\epsilon})$ expected-time randomized algorithm for the problem. Moreover, they showed that the decision version in this case can be solved in linear time.

Another well-known variant of the Fréchet distance is the *weak discrete Fréchet distance* (WDFD), in which the frogs are allowed to jump also backwards to the previous point in their sequence. Alt and Godau [2] showed that the continuous weak Fréchet distance can be computed in $O(mn \log(mn))$ time.

In many applications, the input curves are not necessarily aligned, and one of them needs to be adjusted (i.e., undergo some transformation) for the distance computation to be meaningful. In the *discrete Fréchet distance under translation*, we are given two sequences of points $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_m)$, and wish to find a translation t that minimizes the discrete Fréchet distance between A and $B + t$.

For points in the plane, Alt et al. [3] gave an $O(m^3n^3(m+n)^2 \log(m+n))$-time algorithm for computing the continuous Fréchet distance under translation, and an algorithm computing a $(1+\epsilon)$-approximation in $O(\epsilon^{-2}mn)$ time. In 3D, Wenk [24] showed that the minimum continuous Fréchet distance under any reasonable family of transformations, can be computed in $O((m+n)^{3f+2} \log(m+n))$ time, where f is the number of degrees of freedom for moving one sequence w.r.t. the other. For translations only ($f = 3$), the minimum continuous Fréchet distance in $\mathbb{R}^3$ can be computed in $O((m+n)^{11} \log(m+n))$ time.

In the discrete case, the situation is a little better. For points in the plane, Jiang et al. [20] gave an $O(m^3n^3 \log(m+n))$-time algorithm for DFD under translation, and an $O(m^4n^4 \log(m+n))$-time algorithm when both rotations and translations are allowed. Mosig et al. [22] presented an approximation algorithm for DFD under translation, rotation and scaling in the plane, with approximation factor close to 2 and running time $O(m^2n^2)$. Finally, Ben Avraham et al. [5] presented an $O(m^3n^2(1+\log(n/m)) \log(m+n))$-time algorithm for DFD under translation.

1.1 Preliminaries

Let $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_m)$ be two sequences of points. We define a directed graph $G = G(V = A \times B, E = E_A \cup E_B \cup E_{AB})$, whose vertices are the possible positions of the frogs and whose edges are the possible moves between positions:
$E_A = \{\langle (a_i, b_j), (a_{i+1}, b_j)\rangle\}, E_B = \{\langle (a_i, b_j), (a_i, b_{j+1})\rangle\}, E_{AB} = \{\langle (a_i, b_j), (a_{i+1}, b_{j+1})\rangle\}$.

The set E_A corresponds to moves where only the A-frog jumps forward, the set E_B corresponds to moves where only the B-frog jumps forward, and the set E_{AB} corresponds to moves where both frogs jump forward. Notice that any valid sequence of moves (with unlimited leash length) corresponds to a path in G from (a_1, b_1) to (a_n, b_m), and vice versa.

It is likely that not all positions in $A \times B$ are **valid**; for example, when the leash is short. We thus assume that we are given an **indicator** function $\sigma : A \times B \to \{0, 1\}$, which determines for each position whether it is valid or not. Now, we say that a position (a_i, b_j) is a **reachable position** (w.r.t. σ), if there exists a path P in G from (a_1, b_1) to (a_i, b_j), consisting of only valid positions, i.e., for each position $(a_k, b_l) \in P$, we have $\sigma(a_k, b_l) = 1$.

Let $d(a_i, b_j)$ denote the Euclidean distance between a_i and b_j. For any distance $\delta \geq 0$, the function σ_δ is defined as follows: $\sigma_\delta(a_i, b_j) = \begin{cases} 1, & d(a_i, b_j) \leq \delta \\ 0, & \text{otherwise} \end{cases}$.

The **discrete Fréchet distance** $d_{dF}(A, B)$ is the smallest $\delta \geq 0$ for which (a_n, b_m) is a reachable position w.r.t. σ_δ.

One-sided shortcuts. Let σ be an indicator function. We say that a position (a_i, b_j) is an **s-reachable position** (w.r.t. σ), if there exists a path P in G from (a_1, b_1) to (a_i, b_j), such that $\sigma(a_1, b_1) = 1$, $\sigma(a_i, b_j) = 1$, and for each b_l, $1 < l < j$, there exists a position $(a_k, b_l) \in P$ that is valid (i.e., $\sigma(a_k, b_l) = 1$). We call such a path an **s-path**. In general, an s-path consists of both valid and non-valid positions. Consider the sequence S of positions that is obtained from P by deleting the non-valid positions. Then S corresponds to a sequence of moves, where the A-frog is allowed to skip points, and the leash satisfies σ. Since in any path in G the two indices (of the A-points and of the B-points) are monotonically non-decreasing, it follows that in S the B-frog visits each of the points $b_1, \ldots, b_j$, in order, while the A-frog visits only a subset of the points $a_1, \ldots, a_i$ (including a_1 and a_i), in order.

The **discrete Fréchet distance with shortcuts** $d^s_{dF}(A, B)$ is the smallest $\delta \geq 0$ for which (a_n, b_m) is an s-reachable position w.r.t. σ_δ.

Weak Fréchet distance. Let $G_w = G(V = A \times B, E_w)$, where $E_w = \{(u, v) | \langle u, v\rangle \in E_A \cup E_B \cup E_{AB}\}$. That is, G_w is an undirected graph obtained from the graph G of the 'strong' version, which contains directed edges, by removing the directions from the edges. Let σ be an indicator function. We say that a position (a_i, b_j) is a **w-reachable position** (w.r.t. σ), if there exists a path P in G_w from (a_1, b_1) to (a_i, b_j) consisting of only valid positions. Such a path corresponds to a sequence of moves of the frogs, with a leash satisfying σ, where backtracking is allowed.

The **weak discrete Fréchet distance** $d^w_{dF}(A, B)$ is the smallest $\delta \geq 0$ for which (a_n, b_m) is a w-reachable position w.r.t. σ_δ.

The translation problem. Given two sequences of points $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_m)$, we wish to find a translation t^* that minimizes $d_{dF}(A, B+t)$ (similarly, $d^s_{dF}(A, B+t)$ and $d^w_{dF}(A, B+t)$), over all translations t. Denote $\widehat{d_{dF}}(A, B) = \min_t\{d_{dF}(A, B+t)\}$, $\widehat{d^s_{dF}}(A, B) = \min_t\{d^s_{dF}(A, B+t)\}$ and $\widehat{d^w_{dF}}(A, B) = \min_t\{d^w_{dF}(A, B+t)\}$.

1.2 Our results

As mentioned earlier, Ben Avraham et al. [5] presented an algorithm that computes DFD under translation in $O(m^3n^2(1+\log(n/m))\log(m+n))$ time. Given sequences A and B and an indicator function σ, they construct a dynamic data structure in $O(mn)$ time (which also stores the information whether (a_n, b_m) is reachable or not). Following a single change (i.e., some valid position becomes non-valid or vice versa), the data structure can be updated in $O(m(1+\log(n/m)))$ time.

Our first major result is an efficient algorithm for DFDS under translation. We provide a dynamic data structure which supports updates in $O(\log(m+n))$ time per update, where in an update the value of σ for some position (a_i, b_j) changes from valid to non-valid or vice versa. Following an update, one can determine whether the final position (a_n, b_m) is reachable from the starting position (a_1, b_1), with shortcuts, in $O(\log(m+n))$ time. The data structure is based on Sleator and Tarjan's Link-Cut Trees structure [23], and, by plugging it into the optimization algorithm of Ben Avraham et al. [5], we obtain an $O(m^2n^2\log^2(m+n))$-time algorithm for DFDS under translation; an order of magnitude faster than the the algorithm for DFD under translation.

In 1D, the optimization algorithm of [5] yields an $O(m^2n(1+\log(n/m))\log(m+n))$-time algorithm for DFD, using their reachability structure, an $O(mn\log^2(m+n))$-time algorithm for DFDS, using our reachability with shortcuts structure, and an $O(mn\log^2(m+n))$-time algorithm for WDFD, using a reachability structure of Eppstein et al. [15] for undirected planar graphs. We describe a simpler optimization algorithm for 1D, which avoids the need for parametric search and yields an $O(m^2n(1+\log(n/m)))$-time algorithm for DFD and $O(mn\log(m+n))$-time algorithms for DFDS and WDFD; i.e., we remove a logarithmic factor from the bounds obtained with the algorithm of Ben Avraham et al.

Our optimization algorithm for 1D follows a general scheme introduced by Martello et al. [21] for the Balanced Optimization Problem (BOP). BOP is defined as follows. Let $E = \{e_1, \ldots, e_l\}$ be a set of l elements (where here $l = O(mn)$), $c : E \to \mathbb{R}$ a cost function, and $\mathcal{F}$ a set of feasible subsets of E. Find a feasible subset $S^* \in \mathcal{F}$ that minimizes $\max\{c(e_i) : e_i \in S\} - \min\{c(e_i) : e_i \in S\}$, over all $S \in \mathcal{F}$. Given a feasibility decider that decides whether a subset is feasible or not in $f(l)$ time, the algorithm of [21] finds an optimal range in $O(lf(l) + l\log l)$-time.

The scheme of [21] is especially useful when an efficient dynamic version of the feasibility decider is available, as in the case of DFD (where $f(l) = O(m(1+\log(n/m)))$), DFDS (where $f(l) = O(\log(m+n))$), and WDFD (where $f(l) = O(\log(m+n))$).

Our second major result is an alternative scheme for BOP. Our optimization scheme does not require a specially tailored dynamic version of the feasibility decider in order to obtain faster algorithms (than the naive $O(lf(l) + l\log l)$ one), rather, whenever the underlying problem has some desirable properties, it produces algorithms with running time $O(f(l)\log^2 l + l\log l)$. Thus, the advantage of our scheme is that it yields efficient algorithms quite easily, without having to devise an efficient dynamic version of the feasibility decider, a task which is often difficult if at all possible.

We demonstrate our scheme on the *most uniform path* problem (MUPP). Given a weighted graph $G = (V, E, w)$ with n vertices and m edges and two vertices $s, t \in V$, the goal is to find a path P^* in G between s and t that minimizes $\max\{w(e) : e \in P\} - \min\{w(e) : e \in P\}$, over all paths P from s to t. This problem was introduced by Hansen et al. [18], who gave an $O(m^2)$-time algorithm for it. By using a dynamic connectivity data structure of Holm et al. [19], one can reduce the running time to $O(m\log^2 n)$. We apply our scheme to MUPP to obtain a much simpler algorithm with the same ($O(m\log^2 n)$) running time. Finally, we

observe that WDFD under translation in 1D can be viewed as a special case of MUPP, so we immediately obtain a much simpler algorithm than the one based on Eppstein et al.'s dynamic data structure (see above), at the cost of an additional logarithmic factor.

2 DFDS under translation

The discrete Fréchet distance (and its shortcuts variant) between A and B is determined by two points, one from A and one from B. Consider the decision version of the translation problem: given a distance δ, decide whether $\widehat{d_{dF}}(A,B) \leq \delta$ (or $\widehat{d^s_{dF}}(A,B) \leq \delta$).

Ben Avaraham et al. [5] described a subdivision of the plane of translations: given two points $a \in A$ and $b \in B$, consider the disk $D_\delta(a-b)$ of radius δ centered at $a-b$, and notice that $t \in D_\delta(a-b)$ if and only if $d(a-b,t) \leq \delta$ (or $d(a,b+t) \leq \delta$). That is, $D_\delta(a-b)$ is precisely the set of translations t for which $b+t$ is at distance at most δ from a. They construct the arrangement A_δ of the disks in $\{D_\delta(a-b) \mid (a,b) \in A \times B\}$, which consists of $O(m^2n^2)$ cells. Then, they initialize their dynamic data structure for (discrete Fréchet) reachability queries, and traverse the cells of A_δ such that, when moving from one cell to its neighbor, the dynamic data structure is updated and queried a constant number of times in $O(m(1+\log(n/m))$ time. Finally, they use parametric search in order to find an optimal translation, which adds only a $O(\log(m+n))$ factor to the running time.

In this section we present a dynamic data structure for s-reachability queries, which allows updates and queries in $O(\log(m+n))$ time. We observe that the same parametric search can be used in the shortcuts variant, since the critical values are the same. Thus, by combining our dynamic data structure with the parametric search of [5], we obtain an $O(m^2n^2\log^2(m+n))$-time algorithm for DFDS under translation.

We now describe the dynamic data structure for DFDS. Consider the decision version of the problem: given a distance δ, we would like to determine whether $d^s_{dF}(A,B) \leq \delta$, i.e., whether (a_n,b_m) is an s-reachable position w.r.t. σ_δ. In [4], Ben Avraham et al. presented a linear time algorithm for this decision problem. Informally, the decision algorithm on the graph G is as follows: starting at (a_1,b_1), the B-frog jumps forward (one point at a time) as long as possible, while the A-frog stays in place, then the A-frog makes the smallest forward jump needed to allow the B-frog to continue. They continue advancing in this way, until they either reach (a_n,b_m) or get stuck.

Consider the (directed) graph $G_\delta = G(V = A \times B, E = E'_A \cup E'_B)$, where
$E'_A = \{\langle (a_i,b_j),(a_{i+1},b_j)\rangle \mid \sigma_\delta(a_i,b_j) = 0,\ 1 \leq i \leq n-1,\ 1 \leq j \leq m\}$, and
$E'_B = \{\langle (a_i,b_j),(a_i,b_{j+1})\rangle \mid \sigma_\delta(a_i,b_j) = 1,\ 1 \leq i \leq n,\ 1 \leq j \leq m-1\}$.

In G_δ, if the current position of the frogs is valid, only the B-frog may jump forward and the A-frog stays in place. And, if the current position is non-valid, the B-frog stays in place and only the A-frog may jump forward. Let M_δ be an $n \times m$ matrix such that $M_{i,j} = \sigma_\delta(a_i,b_j)$. Each vertex in G_δ corresponds to a cell of the matrix. The directed edges of G_δ correspond to right-moves (the B-frog jumps forward) and upward-moves (the A-frog jumps forward) in the matrix. Any right-move is an edge originating at a valid vertex, and any upward-move is an edge originating at a non-valid vertex (see Figure 1).

▶ **Observation 1.** *G_δ is a set of rooted binary trees, where a root is a vertex of out-degree 0.*

Proof. Clearly, G is a directed acyclic graph, and G_δ is a subgraph of G. In G_δ, each vertex has at most one outgoing edge. It is easy to see (by induction on the number of vertices) that such a graph is a set of rooted trees. ◀

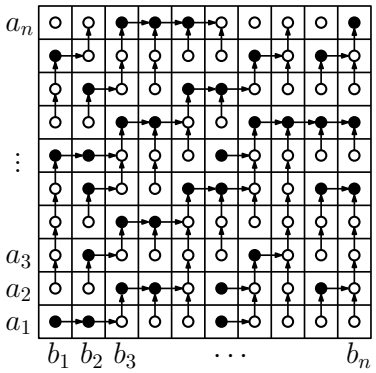

Figure 1 The graph G_δ on the matrix M_δ. The black vertices are valid and the white ones are non-valid.

We call a path P in G from (a_i, b_j) to $(a_{i'}, b_{j'})$, $i \le i'$, $j \le j'$, a **partial s-path**, if for each b_l, $j \le l < j'$, there exists a position $(a_k, b_l) \in P$ that is valid (i.e., $\sigma_\delta(a_k, b_l) = 1$).

▶ **Observation 2.** *All the paths in G_δ are partial s-paths.*

Proof. Let P be a path from (a_i, b_j) to $(a_{i'}, b_{j'})$ in G_δ. Each right-move in P advances the B-frog by one step forward. If $j = j'$ then the claim is vacuously true. Else, P must contain a right-move for each b_l, $j \le l < j'$. Any right-move is an edge originating at a valid vertex, thus for any $j \le l < j'$ there exists a position $(a_k, b_l) \in P$ such that $\sigma_\delta(a_k, b_l) = 1$. ◀

Denote by $r(a_i, b_j)$ the root of (a_i, b_j) in G_δ.

▶ **Lemma 3.** *(a_n, b_m) is an s-reachable position in G w.r.t. σ_δ, if and only if $\sigma_\delta(a_1, b_1) = 1$, $\sigma_\delta(a_n, b_m) = 1$, and $r(a_1, b_1) = (a_i, b_m)$ for some $1 \le i \le n$.*

Proof. Assume that $\sigma_\delta(a_1, b_1) = 1$, $\sigma_\delta(a_n, b_m) = 1$, and $r(a_1, b_1) = (a_i, b_m)$ for some $1 \le i \le n$. Then by Observation 2 there is a partial s-path from (a_1, b_1) to (a_i, b_m) in G_δ, and since $\sigma_\delta(a_1, b_1) = 1$ and $\sigma_\delta(a_n, b_m) = 1$ we have an s-path from (a_1, b_1) to (a_n, b_m).

Now assume that (a_n, b_m) is an s-reachable position in G w.r.t. σ_δ. Then, in particular, $\sigma_\delta(a_1, b_1) = 1$ and $\sigma_\delta(a_n, b_m) = 1$, and there exists an s-path P in G from (a_1, b_1) to (a_n, b_m). Let P' be the path in G_δ from (a_1, b_1) to $r(a_1, b_1)$. Informally, we claim that P' is always not above P. More precisely, we prove that if a position (a_i, b_j) is an s-reachable position in G, then there exists a position $(a_{i'}, b_j) \in P'$, $i' \le i$, such that $\sigma_\delta(a_{i'}, b_j) = 1$. In particular, since (a_n, b_m) is an s-reachable position in G, there exists a position $(a_{i'}, b_m) \in P'$, $i' \le n$, such that $\sigma_\delta(a_{i'}, b_m) = 1$, and thus $r(a_1, b_1) = (a_{i''}, b_m)$ for some $i' \le i'' \le n$.

We prove this claim by induction on j. The base case where $j = 1$ is trivial, since $(a_1, b_1) \in P \cap P'$ and $\sigma_\delta(a_1, b_1) = 1$. Let P be an s-path from (a_1, b_1) to (a_i, b_{j+1}), then $\sigma_\delta(a_i, b_{j+1}) = 1$. Let (a_k, b_j), $k \le i$, be a position in P such that $\sigma_\delta(a_k, b_j) = 1$. (a_k, b_j) is an s-reachable position in G, so by the induction hypothesis there exists a vertex $(a_{k'}, b_j) \in P'$, $k' \le k$, such that $\sigma_\delta(a_{k'}, b_j) = 1$. By the construction of G_δ, there is an edge $\langle (a_{k'}, b_j), (a_{k'}, b_{j+1}) \rangle$, and we have $(a_{k'}, b_{j+1}) \in P'$. Now, let $k' \le i' \le i$ be the smallest index such that $\sigma_\delta(a_{i'}, b_{j+1}) = 1$. Since there are no right-moves in P' before reaching $(a_{i'}, b_{j+1})$, we have $(a_{i'}, b_{j+1}) \in P'$. ◀

We represent G_δ using the Link-Cut tree data structure, which was developed by Sleator and Tarjan [23]. The data structure stores a set of rooted trees and supports the following operations in $O(\log n)$ amortized time:

- $Link(v, u)$ – connect a root node v to another node u as its child.
- $Cut(v)$ – disconnect the subtree rooted at v from the tree to which it belong.
- $FindRoot(v)$ – find the root of the tree to which v belongs.

Now, in order to maintain the representation of G_δ following a single change in σ_δ (i.e., when switching one position (a_i, b_j) from valid to non-valid or vice versa), one edge should be removed and one edge should be added to the structure. We update our structure as follows: Let T be the tree containing (a_i, b_j).

- When switching (a_i, b_j) from valid to non-valid, we first need to remove the edge $\langle (a_i, b_j), (a_i, b_{j+1}) \rangle$, if $j < m$, by disconnecting (a_i, b_j) (and its subtree) from T ($Cut(a_i, b_j)$). Then, if $i < n$, we add the edge $\langle (a_i, b_j), (a_{i+1}, b_j) \rangle$ by connecting (a_i, b_j) (which is now the root of its tree) to (a_{i+1}, b_j) as its child ($Link((a_i, b_j), (a_{i+1}, b_j))$).
- When switching a position from non-valid to valid, we need to remove the edge $\langle (a_i, b_j), (a_{i+1}, b_j) \rangle$, if $i < n$, by disconnecting (a_i, b_j) (and its subtree) from T ($Cut(a_i, b_j)$). Then, if $j < m$, we add the edge $\langle (a_i, b_j), (a_i, b_{j+1}) \rangle$ by connecting (a_i, b_j) (which is now the root of its tree) to (a_i, b_{j+1}) as its child ($Link((a_i, b_j), (a_i, b_{j+1}))$).

Assume $\sigma_\delta(a_1, b_1) = \sigma_\delta(a_n, b_m) = 1$. By Lemma 3, in the Link-Cut tree data structure representing G_δ, $FindRoot(a_1, b_1)$ is (a_i, b_m) for some $1 \leq i \leq n$ if and only if (a_n, b_m) is an s-reachable position in G w.r.t. σ_δ. We thus obtain the following theorem.

▶ **Theorem 4.** *Given sequences A and B and an indicator function σ_δ, one can construct a dynamic data structure in $O(mn \log(m+n))$ time, which supports the following operations in $O(\log(m+n))$ time: (i) change a single value of σ_δ, and (ii) check whether (a_n, b_m) is an s-reachable position in G w.r.t. σ_δ.*

▶ **Theorem 5.** *Given sequences A and B with n and m points respectively in the plane, $\widehat{d^s_{dF}}(A, B)$ can be computed in $O(m^2 n^2 \log^2(m+n))$-time.*

3 Translation in 1D

The algorithm of [5] can be generalized to any constant dimension $d \geq 1$; only the size of the arrangement of balls, A_δ, changes to $O(m^d n^d)$. The running time of the algorithm for two sequences of points in $\mathbb{R}^d$ is therefore $O(m^{d+1} n^d (1 + \log(n/m)) \log(m+n))$, for DFD, and $O(m^d n^d \log^2(m+n))$, for DFDS and WDFD; see relevant paragraph in Section 1.2.

When considering the translation problem in 1D, we can improve the bounds above by a logarithmic factor, by avoiding the use of parametric search and applying a direct approach instead. We thus obtain an $O(m^2 n(1 + \log(n/m)))$-time algorithm, for DFD, and an $O(mn \log(m+n))$-time algorithm, for DFDS and WDFD.

Let $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_m)$ be two sequences of points in $\mathbb{R}^d$. Consider the set $\mathcal{D} = \{a_i - b_j \mid a_i \in A, b_j \in B\}$. Then, each vertex $v = (a_i, b_j)$ of the graph G has a corresponding point $\overline{v} = (a_i - b_j)$ in $\mathcal{D}$. Given a path P in G from (a_1, b_1) to (a_n, b_m), denote by $\overline{V(P)}$ the set of points of $\mathcal{D}$ corresponding to the vertices $V(P)$ of P. Denote by $S(o, r)$ the sphere with center o and radius r. We define a new indicator function:

$$\sigma_{S(o,r)}(a_i, b_j) = \begin{cases} 1, & d(a_i - b_j, o) \leq r \\ 0, & \text{otherwise} \end{cases}.$$

▶ **Lemma 6.** *Let $S = S(t^*, \delta)$ be a smallest sphere for which (a_n, b_m) is a reachable position w.r.t. σ_S. Then, t^* is a translation that minimizes $d_{dF}(A, B + t)$, over all translations t, and $d_{dF}(A, B + t^*) = \delta$.*

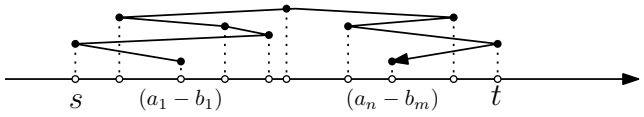

Figure 2 The points of $\overline{V(P)}$.

Proof. Let t be a translation such that $d_{dF}(A, B + t) = \delta'$, and denote $S' = S(t, \delta')$. Thus, there exist a path P from (a_1, b_1) to (a_n, b_m) in G such that for each vertex (a, b) of P, $d(a, b + t) \leq \delta'$. But $d(a, b + t) = d(a - b, t)$, so for each vertex (a, b) of P, $d(a - b, t) \leq \delta'$, and thus (a_n, b_m) is a reachable position w.r.t. $\sigma_{S'}$. Since S is the smallest sphere for which (a_n, b_m) is a reachable position w.r.t. σ_S, we get that $\delta' \geq \delta$.

Now, since (a_n, b_m) is a reachable position w.r.t. σ_S, there exists a path P from (a_1, b_1) to (a_n, b_m), such that for each vertex (a, b) of P, $d(a - b, t^*) \leq \delta$. But again $d(a - b, t^*) = d(a, b + t^*)$, and thus $d_{dF}(A, B + t^*) \leq \delta$. ◀

Notice that the above lemma is true for the shortcuts and the weak variants as well, by letting (a_n, b_m) be an s-reachable or a w-reachable position, respectively.

Thus, our goal is to find the smallest sphere S for which (a_n, b_m) is a reachable position w.r.t. σ_S. We can perform an exhaustive search: check for each sphere S defined by $d + 1$ points of $\mathcal{D}$ whether (a_n, b_m) is a reachable position w.r.t. σ_S. There are $O(m^{d+1}n^{d+1})$ such spheres, and checking whether (a_n, b_m) is a reachable position in G takes $O(mn)$ time. This yields an $O(m^{d+2}n^{d+2})$-time algorithm.

When considering the problem on the line, the goal is to find a path P from (a_1, b_1) to (a_n, b_m), such that the one-dimensional distance between the leftmost point in $\overline{V(P)}$ and the rightmost point in $\overline{V(P)}$ is minimum (see Figure 2). In other words, our indicator function is now defined for a given range $[s, t]$: $\sigma_{[s,t]}(a_i, b_j) = \begin{cases} 1, & s \leq a_i - b_j \leq t \\ 0, & \text{otherwise} \end{cases}$.

We say that a range $[s, t]$ is a **feasible range** if (a_n, b_m) is a reachable position in G w.r.t $\sigma_{[s,t]}$. Now, we need to find the smallest feasible range delimited by two points of $\mathcal{D}$.

Consider the following search procedure: Sort the values in $\mathcal{D} = \{d_1, \ldots, d_l\}$ such that $d_1 < d_2 < \cdots < d_l$, where $l = mn$. Set $p \leftarrow 1, q \leftarrow 1$. While $q \leq l$, if (a_n, b_m) is a reachable position in G w.r.t. $\sigma_{[d_p,d_q]}$, set $p \leftarrow p + 1$, else set $q \leftarrow q + 1$. Return the translation corresponding to the smallest feasible range $[d_p, d_q]$ that was found during the while loop.

We use the data structure of [5] for the decision queries, and update it in $O(m(1+\log(n/m))$ time in each step of the algorithm. For DFDS we use our data structure, and for WDFD we use the data structure of [15], where in both the cost of a decision query or an update is $O(\log(m + n))$.

▶ **Theorem 7.** *Let A and B be two sequences of n and m points ($m \leq n$), respectively, on the line. Then, $\widehat{d_{dF}}(A, B)$ can be computed in $O(m^2 n(1 + \log(n/m)))$ time, and $\widehat{d^s_{dF}}(A, B)$ and $\widehat{d^w_{dF}}(A, B)$ can be computed in $O(mn \log(m + n))$ time.*

4 A general scheme for BOP

In the previous section we showed that DFD, DFDS, and WDFD, all under translation and in 1D, can be viewed as BOP. In this section, we present a general scheme for BOP, which yields efficient algorithms quite easily, without having to devise an efficient dynamic version of the feasibility decider.

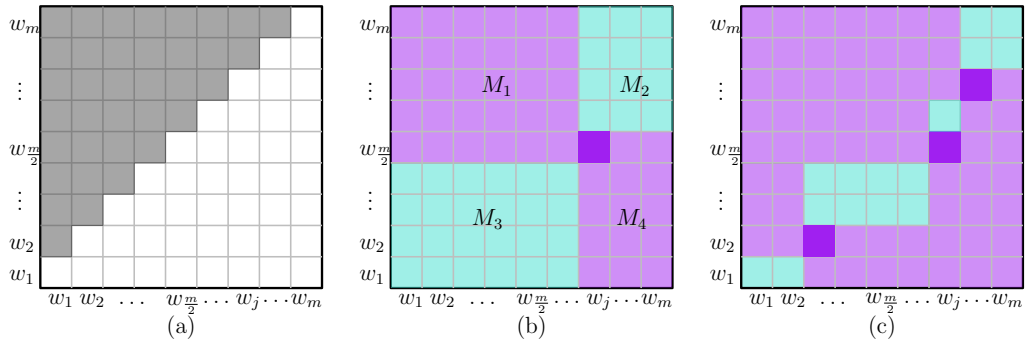

Figure 3 The matrix of possible ranges. (a) The shaded cells are invalid ranges. (b) The cell $M_{\frac{m}{2},j}$ induces a partition of M into 4 submatrices: M_1, M_2, M_3, M_4. (c) The four submatrices at the end of the second level of the recursion tree.

BOP's definition (see Section 1.2) is especially suited for graphs, where, naturally, E is the set of weighted edges of the graph, and $\mathcal{F}$ is a family of well-defined structures, such as matchings, paths, spanning trees, cut-sets, edge covers, etc.

Let $G = (V, E, w)$ be a weighted graph, where V is a set of n vertices, E is a set of m edges, and $w : E \to \mathbb{R}$ is a weight function. Let $\mathcal{F}$ be a set of feasible subsets of E. For a subset $S \subseteq E$, let $S_{max} = \max\{w(e) : e \in S\}$ and $S_{min} = \min\{w(e) : e \in S\}$. The Balanced Optimization Problem on Graphs (BOPG) is to find a feasible subset $S^* \in \mathcal{F}$ which minimizes $S_{max} - S_{min}$ over all $S \in \mathcal{F}$. A range $[l, u]$ is a **feasible range** if there exists a feasible subset $S \in \mathcal{F}$ such that $w(e) \in [l, u]$ for each $e \in S$. A **feasibility decider** is an algorithm that decides whether a given range is feasible.

We assume for simplicity that each edge has a unique weight. Our goal is to find the smallest feasible range. First, we sort the m edges by their weights, and let $e_1, e_2, \ldots, e_m$ be the resulting sequence. Let $w_1 = w(e_1) < w_2 = w(e_2) < \cdots < w_m = w(e_m)$.

Let M be the matrix whose rows correspond to $w_1, w_2, \ldots, w_m$ and whose columns correspond to $w_1, w_2, \ldots, w_m$ (see Figure 3(a)). A cell $M_{i,j}$ of the matrix corresponds to the range $[w_i, w_j]$. Notice that some of the cells of M correspond to invalid ranges: when $i > j$, we have $w_i > w_j$ and thus $[w_i, w_j]$ is not a valid range.

M is sorted in the sense that range $M_{i,j}$ contains all the ranges $M_{i',j'}$ with $i \leq i' \leq j' \leq j$. Thus, we can perform a binary search in the middle row to find the smallest feasible range $M_{\frac{m}{2},j} = [w_{\frac{m}{2}}, w_j]$ among the ranges in this row. $M_{\frac{m}{2},j}$ induces a partition of M into 4 submatrices: M_1, M_2, M_3, M_4 (see Figure 3(b)). Each of the ranges in M_1 is contained in a range of the middle row which is not a feasible range, hence none of the ranges in M_1 is a feasible range. Each of the ranges in M_4 contains $M_{\frac{m}{2},j}$ and hence is at least as large as $M_{\frac{m}{2},j}$. Thus, we may ignore M_1 and M_4 and focus only on the ranges in the submatrices M_2 and M_3.

Sketch of the algorithm. We perform a recursive search in the matrix M. The input to a recursive call is a submatrix M' of M and a corresponding graph G'. Let $[w_i, w_j]$ be a range in M'. The feasibility decider can decide whether $[w_i, w_j]$ is a feasible range or not by consulting the graph G'. In each recursive call, we perform a binary search in the middle row of M' to find the smallest feasible range in this row, using the corresponding graph G'. Then, we construct two new graphs for the two submatrices of M' in which we still need to search in the next level of the recursion.

Algorithm 1 Balance($G([l, l'] \times [u', u])$)

1. Set $i = \frac{l+l'}{2}$
2. Perform a binary search on the ranges $[i, j]$, $u' \leq j \leq u$, to find the smallest feasible range, using the feasibility decider with the graph $G([l, l'] \times [u', u])$ as input.
3. If there is no feasible range, then:
   a. If $l = l'$, return ∞.
   b. Else, construct $G_1 = G([l, i-1] \times [u', u])$ and return Balance(G_1).
4. Else, let $[w_i, w_j]$ be the smallest feasible range found in the binary search.
   a. If $l = l'$, return $(w_j - w_i)$.
   b. Else, construct two new graphs, $G_1 = G([i+1, l'] \times [j, u])$ and $G_2 = G([l, i-1] \times [u', j-1])$, and return $\min\{(w_j - w_i), \text{Balance}(G_1), \text{Balance}(G_2)\}$.

The number of potential feasible ranges is equal to the number of cells in M, which is $O(m^2)$. But, since we are looking for the smallest feasible range, we do not need to generate all of them. We only use M to illustrate the search algorithm, its cells correspond to the potential feasible ranges, but do not contain any values. We thus represent M and its submatrices by the indices of the sorted list of weights that correspond to the rows and columns of M. For example, we represent M by $M([1, m] \times [1, m])$, M_2 by $M([\frac{m}{2} + 1, m] \times [j, m])$, and M_3 by $M([1, \frac{m}{2} - 1] \times [1, j - 1])$. We define the *size* of a submatrix of M by the sum of its number of rows and number of columns, for example, M is of size $2m$, $|M_2| = \frac{3m}{2} - j + 1$, and $|M_3| = \frac{m}{2} + j - 2$.

Each recursive call is associated with a range of rows $[l, l']$ and a range of columns $[u', u]$ (the submatrix $M([l, l'] \times [u', u])$), and a corresponding input graph $G' = G([l, l'] \times [u', u])$. The scheme does not state which edges should be in G' or how to construct it, but it does require the followings properties:

1. The number of edges in G' should be $O(|M'|)$.
2. Given G', the feasibility decider can answer a feasibility query for any range in M', in $O(f(|G'|))$ time.
3. The construction of the graphs for the next level should take $O(|G'|)$ time.

The optimization scheme is given in Algorithm 1; its initial input is $G = G([1, m] \times [1, m])$.

Correctness. Let g be a bivariate real function with the property that for any four values of the weight function $c \leq a \leq b \leq d$, it holds that $g(a, b) \leq g(c, d)$. In our case, $g(a, b) = b - a$. We prove a somewhat more general theorem – that our scheme applies to any such monotone function g; for example, $g(a, b) = b/a$ (assuming the edge weights are positive numbers).

▶ **Theorem 8.** *Algorithm 1 returns the minimum value $g(S_{min}, S_{max})$ over all feasible subsets $S \in \mathcal{F}$.*

Proof. We claim that given a graph $G' = G([l, l'] \times [u', u])$ as input, Algorithm 1 returns the minimal $g(S_{min}, S_{max})$ over all feasible subsets $S \in \mathcal{F}$, such that $S_{min} \in [l, l']$ and $S_{max} \in [u', u]$. Let $M' = M([l, l'] \times [u', u])$ be the corresponding matrix. The proof is by induction on the number of rows in M'.

First, notice that the algorithm runs the feasibility decider only on ranges from M'. The base case is when M' contains a single row, i.e. $l = l'$. In this case the algorithm returns the minimal feasible range $[w_l, w_j]$ such that $j \in [u', u]$, or returns ∞ if there is no such range. Else, M' has more than one row. Assume that there is no feasible range in the middle row

of M'. In other words, there is no $j \in [u', u]$ such that $[w_i, w_j]$ is a feasible range. Trivially, for any $i' > i$ we have $w_{i'} > w_i$, and therefore for any $j \in [u', u]$, $[w_{i'}, w_j]$ is not a feasible range, and the algorithm continues recursively with $G_1 = G([l, i-1] \times [u', u])$. Now assume that $[w_i, w_j]$ is the minimal feasible range in the middle row. We can partition the ranges in M' to four types (submatrices):

1. All the ranges $[w_{i'}, w_{j'}]$ where $i' \in [i+1, l']$ and $j' \in [j, u]$.
2. All the ranges $[w_{i'}, w_{j'}]$ where $i' \in [l, i-1]$ and $j' \in [u', j-1]$.
3. All the ranges $[w_{i'}, w_{j'}]$ where $i' \in [i, l']$ and $j' \in [u', j-1]$. For any such valid range ($j' > i'$), we have $[w_{i'}, w_{j'}] \subseteq [w_i, w_j]$, so it is not a feasible range (otherwise, the result of the binary search would be $[w_i, w_{j'}]$).
4. All the ranges $[w_{i'}, w_{j'}]$ where $i' \in [l, i]$ and $j' \in [j, u]$. Since $j \geq i$, all these ranges are valid. For any such range, we have $w_{i'} \leq w_i \leq w_j \leq w_{j'}$, therefore, all these ranges are feasible, but since $g(w_i, w_j) \leq g(w_{i'}, w_{j'})$, there is no need to check them.

Indeed, the algorithm continues recursively with G_1 and G_2 (corresponding to ranges of type 1 and 2, respectively), which may contain smaller feasible ranges. By the induction hypothesis, the recursive calls return the minimal $g(S_{min}, S_{max})$ over all feasible subsets $S \in \mathcal{F}$, such that $S_{min} \in [i+1, l']$ and $S_{max} \in [j, u]$ or $S_{min} \in [l, i-1]$ and $S_{max} \in [u', j-1]$. Finally, the algorithm returns the minimum over all the feasible ranges in M'. ◀

▶ **Lemma 9.** *The total size of the matrices in each level of the recursion tree is at most* $2m$.

Proof. By induction on the level. The only matrix in level 0 is M, and $|M| = 2m$. Let $M' = M([l, l'] \times [u', u])$ be a matrix in level $i-1$. The size of M' is $l' - l + u - u' + 2$ (it has $l' - l + 1$ rows and $u - u' + 1$ columns). In level i we perform a binary search in the middle row of M' to find the smallest feasible range $[w_{\frac{l+l'}{2}}, w_j]$ in this row. It is easy to see that the resulting two submatrices are of sizes $l' - \frac{l+l'}{2} + u - j + 1$ and $\frac{l+l'}{2} - l + j - u'$, respectively, which sums to $l' - l + u - u' + 1$. ◀

Running time. Consider the recursion tree. It consists of $O(\log m)$ levels, where the i'th level is associated with 2^i disjoint submatrices of M. Level 0 is associated with the matrix $M_0 = M$, level 1 is associated with the submatrices M_2 and M_3 of M (see Figure 3), etc.

In the i'th level we apply Algorithm 1 to each of the 2^i submatrices associated with this level. Let $\{M_k^i\}_{k=1}^{2^i}$ be the submatrices associated with the i'th level. Let G_k^i be the graph corresponding to M_k^i. The size of G_k^i is linear in the size of M_k^i. The feasibility decider runs in $O(f(|M_k^i|))$ time, and thus the binary search in M_k^i runs in $O(f(|M_k^i|) \log |M_k^i|)$ time. Constructing the graphs for the next level takes $O(|M_k^i|)$ time. By lemma 9, the total time spent on the i'th level is $O(\sum_{k=1}^{2^i}(|M_k^i| + f(|M_k^i|) \log |M_k^i|)) \leq O(\sum_{k=1}^{2^i} |M_k^i| + \sum_{k=1}^{2^i} f(|M_k^i|) \log m) = O(m + \log m \sum_{k=1}^{2^i} f(|M_k^i|))$. Finally, the running time of the entire algorithm is $O(m \log m + \sum_{i=1}^{\log m}(m + \log m \sum_{k=1}^{2^i} f(|M_k^i|))) = O(m \log m + \log m \sum_{i=1}^{\log m} \sum_{k=1}^{2^i} f(|M_k^i|))$.

Notice that the number of potential ranges is $O(m^2)$, while the number of weights is only $O(m)$. Nevertheless, whenever $f(|M'|)$ is a linear function, our optimization scheme runs in $O(m \log^2 m)$ time. More generally, whenever $f(|M'|)$ is a function for which $f(x_1) + \cdots + f(x_k) = O(f(x_1 + \cdots + x_k))$, for any $x_1, \ldots, x_k$, our scheme runs in $O(m \log m + f(2m) \log^2 m)$ time.

5 MUPP and WDFD under translation in 1D

In Section 3 we described an algorithm for WDFD under translation in 1D, which uses a dynamic data structure due to Eppstein et al. [15]. In this section we present a much simpler algorithm for the problem, which avoids heavy tools and has roughly the same running time.

As shown in Section 3, WDFD under translation in 1D can be viewed as BOP. More precisely, we say that a range $[s,t]$ is a feasible range if (a_n, b_m) is a w-reachable position in G_w w.r.t. $\sigma_{[s,t]}$. Now, our goal is to find a feasible range of minimum size.

Consider the following weighted graph $\tilde{G}_w = (\tilde{V}_w, \tilde{E}_w, \omega)$, where $\tilde{V}_w = (A \times B) \cup \{v_e \mid e \in E_w\}$, $\tilde{E}_w = \{(u, v_e), (v_e, v) \mid e = (u, v) \in E_w\}$, and $\omega(((a_i, b_j), v_e)) = a_i - b_j$. In other words, $\tilde{G}_w$ is obtained from G_w by adding, for each edge $e = (u, v)$ of G_w, a new vertex v_e, which splits the edge into two new edges, $(u, v_e), (v_e, v)$, whose weight is the distance associated with their original vertex.

Now (a_n, b_m) is a w-reachable position in G_w w.r.t. $\sigma_{[s,t]}$, if and only if there exists a path P between (a_1, b_1) and (a_n, b_m) in G_w such that $\overline{V(P)} \in [s,t]$, if and only if there exists a path $\tilde{P}$ between (a_1, b_1) and (a_n, b_m) in $\tilde{G}_w$ such that for each edge $e \in \tilde{P}$, $\omega(e) \in [s,t]$.

We have reduced our problem to a special case of the most uniform path problem (MUPP). We show below how to apply our scheme to MUPP, with a linear-time feasibility decider, and thus obtain the following theorem as a by-product:

▶ **Theorem 10.** *Let $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_m)$ be two sequences of points in 1D. Then, the weak discrete Fréchet distance under translation, $\widehat{d^w_{dF}}(A, B)$, can be computed in $O(mn \log^2(m+n))$ time.*

Most uniform path. Given a weighted graph $G = (V, E, w)$ with n vertices and m edges, and two vertices $s, t \in V$, the goal is to find a path P^* in G between s and t, which minimizes $\max\{w(e) : e \in P\} - \min\{w(e) : e \in P\}$, over all paths P between s and t.

Here $\mathcal{F}$ is the set of paths in G between s and t. The matrix for the initial call is M and G is its associated graph. Consider a recursive call, and let M' be the submatrix and G' the graph associated with it. Throughout the execution of the algorithm, we maintain the following properties: (i) The number of edges and vertices in G' is at most $O(|M'|)$, and (ii) Given a range $[w_p, w_q]$ in M', there exists a path between s and t in G' with edges in the range $[w_p, w_q]$ if and only if such a path exists in G.

Construction of the graphs for the next level. Given the input graph G' and a submatrix $M'' = M([p, p'] \times [q', q])$ of M', we construct the corresponding graph G'' as follows: First, we remove from G' all the edges e such that $w(e) \notin [w_p, w_q]$. Then, we contract edges with weights in the range $(w_{p'}, w_{q'})$, and finally we remove all the isolated vertices. Notice that G'' is a graph minor of G', and, clearly, all the properties hold.

The feasibility decider. Let $[w_p, w_q]$ be a range from M'. Run a BFS in G', beginning from s, while ignoring edges with weights outside the range $[w_p, w_q]$. If the BFS finds t, return "yes", otherwise return "no". The algorithm returns "yes" if and only if there exists a path between s and t in G' with edges in the range $[w_p, w_q]$, i.e., if and only if such a path exists in G. The running time of the decider is $O(|G'|) = O(|M'|)$.

▶ **Theorem 11.** *The most uniform path problem in G can be solved in $O(m \log^2 n)$ time.*

▶ Remark. We have introduced an alternative optimization scheme for BOP and demonstrated its power. It would be interesting to find additional applications of this scheme. For example, using it we easily obtain an $O(m \log^2 n)$-time algorithm for the *Most Uniform Spanning Tree* problem; slower than the specialized algorithm of Galil and Schieber [17] by only a log-factor.

6 Discussion

In an unpublished manuscript [16], we suggested a new variant of DFD – the *discrete Fréchet gap*. Given two sequences of points $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_n)$, the discrete Fréchet gap between them is the smallest range $[s, t]$, $s \geq t \geq 0$, for which (a_n, b_m) is a reachable position w.r.t. $\sigma_{[s,t]}$, where $\sigma_{[s,t]}(a_i, b_j) = 1$ if and only if $d(a_i, b_j) \in [s, t]$. We used a less general version of our scheme for BOP to solve two variants of the gap problem: the *discrete Fréchet gap with shortcuts* (where (a_n, b_m) is an s-reachable position), and the *weak discrete Fréchet gap* (where (a_n, b_m) is a w-reachable position).

It is interesting to note that DFDS and WDFD, both in 1D under translation, are in some sense analogous to their respective gap variants (in d dimensions and no translation): We can use similar algorithms to compute them, but with different indicator functions. This connection supports the intuition that there is some connection between the discrete Fréchet gap and DFD under translation.

References

1 Pankaj K. Agarwal, Rinat Ben Avraham, Haim Kaplan, and Micha Sharir. Computing the discrete fréchet distance in subquadratic time. *SIAM J. Comput.*, 43(2):429–449, 2014. doi:10.1137/130920526.

2 Helmut Alt and Michael Godau. Computing the fréchet distance between two polygonal curves. *Int. J. Comput. Geometry Appl.*, 5:75–91, 1995. doi:10.1142/S0218195995000064.

3 Helmut Alt, Christian Knauer, and Carola Wenk. Matching polygonal curves with respect to the fréchet distance. In Afonso Ferreira and Horst Reichel, editors, *STACS 2001, 18th Annual Symposium on Theoretical Aspects of Computer Science, Dresden, Germany, February 15-17, 2001, Proceedings*, volume 2010 of *Lecture Notes in Computer Science*, pages 63–74. Springer, 2001. doi:10.1007/3-540-44693-1_6.

4 Rinat Ben Avraham, Omrit Filtser, Haim Kaplan, Matthew J. Katz, and Micha Sharir. The discrete fréchet distance with shortcuts via approximate distance counting and selection. In Siu-Wing Cheng and Olivier Devillers, editors, *30th Annual Symposium on Computational Geometry, SOCG'14, Kyoto, Japan, June 08 - 11, 2014*, page 377. ACM, 2014. doi:10.1145/2582112.2582155.

5 Rinat Ben Avraham, Haim Kaplan, and Micha Sharir. A faster algorithm for the discrete fréchet distance under translation. *CoRR*, abs/1501.03724, 2015. arXiv:1501.03724.

6 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. URL: http://www.vldb.org/archives/website/2005/program/paper/fri/p853-brakatsoulas.pdf.

7 Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 661–670. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.76.

8 Kevin Buchin, Maike Buchin, Wouter Meulemans, and Wolfgang Mulzer. Four soviets walk the dog - with an application to alt's conjecture. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1399–1413. SIAM, 2014. doi:10.1137/1.9781611973402.103.

9 Kevin Buchin, Maike Buchin, and Yusu Wang. Exact algorithms for partial curve matching via the fréchet distance. In Claire Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 645–654. SIAM, 2009. URL: http://dl.acm.org/citation.cfm?id=1496770.1496841.

10 Maike Buchin. *On the computability of the Fréchet distance between triangulated surfaces.* PhD thesis, FU Berlin, 2007.

11 Maike Buchin, Anne Driemel, and Bettina Speckmann. Computing the fréchet distance with shortcuts is np-hard. In Siu-Wing Cheng and Olivier Devillers, editors, *30th Annual Symposium on Computational Geometry, SOCG'14, Kyoto, Japan, June 08 - 11, 2014*, page 367. ACM, 2014. doi:10.1145/2582112.2582144.

12 Anne Driemel and Sariel Har-Peled. Jaywalking your dog: Computing the fréchet distance with shortcuts. *SIAM J. Comput.*, 42(5):1830–1866, 2013. doi:10.1137/120865112.

13 Alon Efrat, Quanfu Fan, and Suresh Venkatasubramanian. Curve matching, time warping, and light fields: New algorithms for computing similarity between curves. *Journal of Mathematical Imaging and Vision*, 27(3):203–216, 2007. doi:10.1007/s10851-006-0647-0.

14 Thomas Eiter and Heikki Mannila. Computing discrete Fréchet distance. Technical Report CD-TR 94/64, Information Systems Dept., Technical University of Vienna, 1994.

15 David Eppstein, Giuseppe F. Italiano, Roberto Tamassia, Robert Endre Tarjan, Jeffery Westbrook, and Moti Yung. Maintenance of a minimum spanning forest in a dynamic plane graph. *J. Algorithms*, 13(1):33–54, 1992. doi:10.1016/0196-6774(92)90004-V.

16 Omrit Filtser and Matthew J. Katz. The discrete fréchet gap. *CoRR*, abs/1506.04861, 2015. arXiv:1506.04861.

17 Zvi Galil and Baruch Schieber. On finding most uniform spanning trees. *Discrete Applied Mathematics*, 20(2):173–175, 1988. doi:10.1016/0166-218X(88)90062-5.

18 Pierre Hansen, Giovanni Storchi, and Tsevi Vovor. Paths with minimum range and ratio of arc lengths. *Discrete Applied Mathematics*, 78(1-3):89–102, 1997. doi:10.1016/S0166-218X(97)00008-5.

19 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001. doi:10.1145/502090.502095.

20 Minghui Jiang, Ying Xu, and Binhai Zhu. Protein structure-structure alignment with discrete fréchet distance. *J. Bioinformatics and Computational Biology*, 6(1):51–64, 2008. doi:10.1142/S0219720008003278.

21 Silvano Martello, WR Pulleyblank, Paolo Toth, and Dominique De Werra. Balanced optimization problems. *Operations Research Letters*, 3(5):275–278, 1984.

22 Axel Mosig and Michael Clausen. Approximately matching polygonal curves with respect to the fre'chet distance. *Comput. Geom.*, 30(2):113–127, 2005. doi:10.1016/j.comgeo.2004.05.004.

23 Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983. doi:10.1016/0022-0000(83)90006-5.

24 Carola Wenk. *Shape matching in higher dimensions.* PhD thesis, Free University of Berlin, Dahlem, Germany, 2003. URL: http://www.diss.fu-berlin.de/2003/151/index.html.