Four Soviets walk the dog, with an application to Alt's conjecture

Four Soviets Walk the Dog—with an Application to Alt's Conjecture

Kevin Buchin *Maike Buchin* *Wouter Meulemans* *Wolfgang Mulzer*

Abstract

Given two polygonal curves in the plane, there are several ways to define a measure of similarity between them. One measure that has been extremely popular in the past is the Fréchet distance. Since it has been proposed by Alt and Godau in 1992, many variants and extensions have been described. However, even 20 years later, the original $O(n^2 \log n)$ algorithm by Alt and Godau for computing the Fréchet distance remains the state of the art (here n denotes the number of vertices on each curve). This has led Helmut Alt to conjecture that the associated decision problem is 3SUM-hard.

In recent work, Agarwal et al. show how to break the quadratic barrier for the *discrete* version of the Fréchet distance, where we consider sequences of points instead of polygonal curves. Building on their work, we give an algorithm to compute the Fréchet distance between two polygonal curves in time $O(n^2\sqrt{\log n}(\log\log n)^{3/2})$ on a pointer machine and in time $O(n^2(\log\log n)^2)$ on a word RAM. Furthermore, we show that there exists an algebraic decision tree for the Fréchet problem of depth $O(n^{2-\varepsilon})$, for some $\varepsilon > 0$. This provides evidence that computing the Fréchet distance may not be 3SUM-hard after all and reveals an intriguing new aspect of this well-studied problem.

1 Introduction

Shape matching is a fundamental problem in computational geometry, computer vision, and image processing. A simple version of the problem can be stated as follows: given a database $\mathcal{D}$ of shapes (or images) and a query shape S, find the shape in $\mathcal{D}$ that most resembles S. Before we can solve this problem, however, we first need to address a much more fundamental issue: what does it mean for two shapes to be similar? In the mathematical literature, there are many different notions of distance between two sets, a prominent example being the *Hausdorff distance*.

The Hausdorff distance has the advantage that it is easy to describe and to compute. However, the *Fréchet distance* better captures the similarity of shapes as perceived by human observers [4], since unlike the Hausdorff distance, it takes the continuity of the shapes into account. This motivated Alt and Godau to study the Fréchet distance, and they describe an $O(n^2 \log n)$ time algorithm to compute it on a real RAM/pointer machine [6]. Since their seminal paper,

TU Eindhoven, Netherlands, {k.a.buchin, m.e.buchin, w.meulemans}@tue.nl. M. Buchin supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 612.001.106. W. Meulemans supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 639.022.707. K. Buchin and M. Buchin supported in part by COST (European Cooperation in Science and Technology) ICT Action IC0903 MOVE.

Freie Universität Berlin, Germany, mulzer@inf.fu-berlin.de. Supported in part by DFG project MU/3501/1.

there has been a wealth of research in various directions, such as extensions to higher dimensions [5, 17, 19, 21, 24, 32], approximation algorithms [7, 8, 29], locally correct matchings [16], and much more [2, 18, 22, 25, 26, 28, 34, 36–38]. The Fréchet distance and its variants like dynamic time-warping [10] have been used in various applications, with recent work particularly focusing on geographic applications such as map-matching tracking data [11, 41] and moving objects analysis [13, 14, 33].

Despite the reams of published research, the original algorithm by Alt and Godau has not been improved, and the quadratic barrier on the running time of the decision problem remains unbroken. If despite many efforts we fail to improve on a quadratic running time bound for a geometric problem, a possible culprit may be the underlying 3SUM-hardness [31]. This situation induced Helmut Alt to make the following conjecture [4].

Conjecture 1.1 (**Alt's Conjecture**). *Let P and Q be two polygonal curves in the plane. Then it is* 3SUM*-hard to decide whether the Fréchet distance between P and Q is at most* 1.

Here, 1 is an arbitrary constant, which by scaling the curves can be changed to any other bound. The only non-trivial known lower bound known for this problem is an $\Omega(n \log n)$ bound in the algebraic computation tree model [15].

Recently, Agarwal et al. [1] showed how to achieve a subquadratic running time for the *discrete* version of the Fréchet distance. Their approach is based on precomputing small parts of the solution. We follow a similar approach based on the so-called Four-Russian-trick which precomputes small recurring parts of the solution and uses table-lookup to speed up the whole computation.[1] Their result is stated in the word RAM model of computation. They ask whether their result can be generalized to the case of the original (continuous) Fréchet distance.

Our contribution. We address the question by Agarwal et al. and show how to extend their approach in order to compute the Fréchet distance between two polygonal curves in total time $O(n^2\sqrt{\log n}(\log\log n)^{3/2})$. This is the first algorithm to achieve a running time of $o(n^2 \log n)$ and constitutes the first improvement for the general case since the original paper by Alt and Godau [6]. We emphasize that our algorithm runs on a real RAM/pointer machine and does not require any bit manipulation tricks to achieve the speedup. If we relax the model to allow constant time table-lookups, the running time can be improved to be almost quadratic, up to $O(\log\log n)$ factors. As in Agarwal et al., our results are achieved by first giving a faster algorithm for the decision version, and then performing an appropriate search over the critical values to solve the optimization problem.

In addition, we show that *non-uniformly*, the Fréchet distance can be computed in subquadratic time. More precisely, we prove that there exists an algebraic decision tree [9] for the decision version of the problem with depth $O(n^{2-\varepsilon})$ for some fixed $\varepsilon > 0$. This makes it unlikely that the Fréchet distance is 3SUM-hard, since it is conjectured that no such decision tree exists for 3SUM [3]. However, it is not clear how to implement this decision tree in subquadratic time, which hints at a discrepancy between the decision tree and the uniform complexity of the Fréchet problem and puts into the illustrious company of such notorious problems as Sorting $X + Y$ [30], Min-Plus-Convolution [12], or finding the Delaunay triangulation for a point set that has been sorted in two orthogonal directions [20]. We find that this aspect of the Fréchet distance is highly intriguing and deserves further study.

[1] Since it is well known that the four Russians are not actually Russian, we refer to them as four Soviets in the title.

2 Preliminaries and Basic Definitions

Let P and Q be two polygonal curves in the plane, defined by their vertices $p_0, p_1, \ldots, p_n$ and $q_0, q_1, \ldots, q_n$.[2] Depending on the context, we will interpret P and Q either as sequences of n edges, or as continuous functions $P, Q : [0, n] \to \mathbb{R}^2$. In the latter case, we have $P(i+\lambda) = (1-\lambda)p_i + \lambda p_{i+1}$ for $i = 0, \ldots, n-1$ and $\lambda \in [0, 1]$, and similarly for Q. Let Ψ be the set of all continuous and nondecreasing functions $\sigma : [0, n] \to [0, n]$ with $\sigma(0) = 0$ and $\sigma(n) = n$. The *Fréchet distance* between P and Q is defined as

$$d_F(P, Q) := \inf_{\sigma \in \Psi} \max_{x \in [0,n]} \|P(x) - Q(\sigma(x))\|,$$

where $\|\cdot\|$ denotes the usual Euclidean distance.

The free-space diagram. The classic approach to compute $d_F(P, Q)$ uses the *free-space diagram* $\mathrm{FSD}(P, Q)$. It is defined as

$$\mathrm{FSD}(P, Q) := \{(x, y) \in [0, n]^2 \mid \|P(x) - Q(y)\| \leq 1\},$$

i.e., $\mathrm{FSD}(P, Q)$ is the subset of the joint parameter space for P and Q where the corresponding points on the curves have distance at most 1. The structure of $\mathrm{FSD}(P, Q)$ is easy to describe: let $R := [0, n]^2$ be the ground set. We subdivide R into n^2 *cells* $C(i, j) = [i-1, i] \times [j-1, j]$, for $i, j = 1, \ldots, n$. The cell $C(i, j)$ corresponds to the edge pair e_i and f_j, where e_i is the ith edge of P and f_j is the jth edge of Q. Then $\mathrm{FSD}(P, Q) \cap C(i, j)$ represents all pairs of points on $e_i \times f_j$ with distance at most 1. Elementary geometry shows that $\mathrm{FSD}(P, Q) \cap C(i, j)$ is the intersection of $C(i, j)$ with an ellipse. In particular, $\mathrm{FSD}(P, Q) \cap C(i, j)$ is convex, and the intersection of $\mathrm{FSD}(P, Q)$ with the boundary of $C(i, j)$ consists of four (possibly empty) intervals, one on each side of $\partial C(i, j)$. We call these intervals the *doors* of $C(i, j)$ in $\mathrm{FSD}(P, Q)$.

A path in $\mathrm{FSD}(P, Q)$ is *bimonotone* if it is both x- and y-monotone. Alt and Godau observed that it suffices to determine whether there exists a bimonotone path from $(0, 0)$ to (n, n) inside $\mathrm{FSD}(P, Q)$. More precisely, set

$$\mathrm{reach}(P, Q) := \{p \in \mathrm{FSD}(P, Q) \mid \ p \text{ is reachable from } (0, 0) \text{ in } \mathrm{FSD}(P, Q) \text{ on a bimonotone path.}\}$$

Then $d_F(P, Q) \leq 1$ if and only if $(n, n) \in \mathrm{reach}(P, Q)$. It is not necessary to compute all of $\mathrm{reach}(P, Q)$: since $\mathrm{FSD}(P, Q)$ is convex inside each cell, we actually just need the intersections $\mathrm{reach}(P, Q) \cap \partial C(i, j)$. These intersections are subintervals of the doors of the free-space diagram, and they can be found in $O(n^2)$ time through a simple traversal of the cells [6].

3 Building a lookup table.

Before even considering the input, our algorithm builds a lookup table. The purpose of this table is to speed up the computation of small parts of the free-space diagram.

Let τ be a parameter, and let $S = [0, \tau]^2$. (A preview for the impatient reader: we will later set $\tau = \sqrt{\log n / \log \log n}$). As in the free-space diagram, we subdivide S into τ^2 cells $D(i, j) :=$

[2] For simplicity, we assume that both curves have the same number of vertices. This constitutes the most interesting case from a theoretical point of view, and it is straightforward to extend our results to the case when the number of vertices on the two curves differs.

$[i-1,i] \times [j-1,j]$, for $i,j = 1,\ldots,\tau$. We denote the left side of the boundary $\partial D(i,j)$ by $l(i,j)$ and the bottom side by $b(i,j)$. Note that $l(i,j)$ coincides with the right side of $\partial D(i-1,j)$ and that $b(i,j)$ coincides with the top of $\partial D(i,j-1)$. Thus, we write $l(\tau+1,j)$ for the *right* side of $D(\tau,j)$ and $b(i,\tau+1)$ for the *top* side of $D(i,\tau)$, for $i,j = 1,\ldots,\tau$. We call the subdivision of S into cells the *elementary grid* G. The *rows* and *columns* of G are defined in the obvious way.

Next, we define a *signature* for the elementary grid. Let $Z = \{s_1,t_1,\ldots,s_{\tau+1},t_{\tau+1}\}$, and let S_Z be the set of all permutations of Z. A signature for G consists of 2τ permutations from S_Z, one permutation σ_i^c for each column i of G and one permutation σ_j^r for each row j of G. We write $x <_i^c y$ if x comes before y in σ_i^c, and $x <_i^r y$ if x comes before y in σ_i^r.

The signature encodes the combinatorial structure of the doors in $\mathrm{FSD}(P,Q)$. Consider a row j. The element s_i represents the lower endpoint of the door on $l(i,j)$, and t_i represents the upper endpoint. The elements s_1,t_1 are an exception: they describe the interval in which $\mathrm{reach}(P,Q)$ intersects $l(1,j)$. The permutation σ_j^r then gives the relative order of the door endpoints. If $t_i <_j^r s_i$, this means that the ith door is closed, i.e., $l(i,j)$ is disjoint from $\mathrm{FSD}(P,Q)$.

Given a signature, we can determine which door boundaries define the intervals in which $\mathrm{reach}(P,Q)$ intersects the edges $l(\tau+1,j)$ and $b(i,\tau+1)$. This is done as follows: for each vertical edge $l(i,j)$ we define a variable $u(i,j)$, and for each horizontal edge $b(i,j)$ we define a variable $v(i,j)$. The $u(i,j)$ are pairs of the form (s_q,t_r), meaning that $\mathrm{reach}(P,Q) \cap l(i,j)$ is bounded by the lower endpoint of the door on $l(q,j)$ and the upper endpoint of the door on $l(r,j)$, where again s_1 and t_1 are special and actually represent the interval $\mathrm{reach}(P,Q) \cap l(1,j)$. If $t_r <_j^r s_q$, then $l(i,j)$ is not reachable. The variable $v(i,j)$ are defined analogously.

The $u(i,j)$ and $v(i,j)$ can be computed recursively as follows: first, we set $u(1,j) = v(i,1) = (s_1,t_1)$ for $i,j = 1,\ldots,\tau$. Next, we describe how to find $u(i,j)$ given $u(i-1,j)$ and $v(i-1,j)$. Write $u(i-1,j) = (s_u,t_u)$ and $v(i-1,j) = (s_v,t_v)$. First suppose $s_v <_{i-1}^c t_v$. This means that $b(i-1,j)$ intersects $\mathrm{reach}(P,Q)$, so $\mathrm{reach}(P,Q) \cap l(i,j)$ is only limited by the door on $l(i,j)$, and we can set $u(i,j) = (s_i,t_i)$. On the other hand, if $s_v >_{i-1}^c t_v$, we cannot cross $b(i,j)$, but must come through $l(i-1,j)$. If $t_u < s_u$ holds, we also cannot come through $l(i-1,j)$, and therefore also set $u(i,j) = (s_u,t_u)$. Otherwise, we will need to pass $l(i,j)$ above s_u and s_i and below t_i, and therefore set $u(i,j) = (\max(s_u,s_i),t_i)$, where the maximum is taken according to the order $<_i^r$. The recursion for the variable $v(i,j)$ is defined similarly.

Now it is easy to see that we can find $u(\tau+1,j)$ and $v(i,\tau+1)$ in total time $O(\tau^2)$, for any given signature. Thus, we enumerate all possible $((2\tau)!)^{2\tau}$ signatures and store the results in a lookup table. This takes total time $O(((2\tau)!)^{2\tau}\tau^2) = \tau^{O(\tau^2)}$.

4 Preprocessing for a given input

Next, we perform a second preprocessing phase that actually considers the input curves P and Q. Our eventual goal is to compute the intersection of $\mathrm{reach}(P,Q)$ with the cell boundaries, while taking advantage of our lookup tables. For this, we subdivide the cells of $\mathrm{FSD}(P,Q)$ into elementary boxes of size τ^2, in the obvious way. We can ignore rounding issues by either duplicating vertices or handling a small part of $\mathrm{FSD}(P,Q)$ without lookup tables.

Eventually we need to determine the signature for each elementary box S. This is not quite possible yet, because the full signature of S depends on the intersection of $\mathrm{reach}(P,Q)$ with the lower and left boundary of S. However, we can find a *partial* signature, in which the position of s_1,t_1 in the permutations σ_i^r, σ_j^c is still to be determined.

We subdivide the columns of $\mathrm{FSD}(P, Q)$ into vertical *strips*, consisting of τ columns each. Let A be such a strip. It corresponds to a subcurve P' of P with τ edges. Let s be a line segment of Q. The *partial signature* of s is the permutation of $Z \setminus \{s_1, t_1\}$ that is induced by the doors of $\mathrm{FSD}(P, Q)$ in the row for s inside A.

Lemma 4.1. *There exists a constant c, such that the following holds: given a subcurve P' with τ vertices, we can compute in $O(\tau^c)$ time a data structure that requires $O(\tau^c)$ space and can determine the partial signature of any line segment on Q in time $O(\log \tau)$.*

Proof. Consider the arrangement $\mathcal{A}$ of unit circles whose centers are the points in P'. The partial signature of a line segment s is determined by the intersections of ℓ_s with the arcs of $\mathcal{A}$ (and for a circle not intersecting s by whether s lies inside or outside of the circle). Let ℓ_s be the line spanned by line segment s. Let ℓ_{sa} be the line parallel to ℓ_s that lies above ℓ_s and has distance 1 from ℓ_s. Let ℓ_{sb} be defined similarly, but below ℓ_s. Suppose we wiggle ℓ_s. Then the order of intersections of ℓ_{sa} and the arcs of $\mathcal{A}$ only changes when ℓ_s moves over a vertex of $\mathcal{A}$ or if ℓ_s leaves or enters a circle. The latter case corresponds to ℓ_{sa} or ℓ_{sb} moving over the center of a circle.

Let V be the set of all vertices of $\mathcal{A}$, and let $V' := V \cup P'$. We compute the arrangement $\mathcal{B}$ of the lines dual to V'. The arrangement $\mathcal{B}$ has $O(\tau^4)$ vertices. We build a point location structure for $\mathcal{B}$ that is very similar to the structure by Dobkin and Lipton [27]. For this, we subdivide $\mathcal{B}$ into strips by drawing a vertical line through each vertex of $\mathcal{B}$. Call the resulting *strip subdivision* $\mathcal{S}$. There are $O(\tau^4)$ strips, and each strip has $O(\tau^2)$ cells.

Now, consider the triple $\phi(s)$ of cells in $\mathcal{S}$ that contain the dual points ℓ_s^*, ℓ_{sa}^*, and ℓ_{sb}^*. The cells in $\phi(s)$ lie in the same strip, and they completely determine the combinatorial structure of the intersection between ℓ_s and $\mathcal{A}$. Thus, for every triple ϕ of cells in a strip of $\mathcal{S}$, we construct a list L_ϕ that represents the combinatorial structure of $\ell_s \cap \mathcal{A}$. There are $O(\tau^4 \cdot \tau^6) = O(\tau^{10})$ such lists, each having size $O(\tau)$. We can compute L_ϕ by traversing the zone of ℓ_s in $\mathcal{A}$. Since unit circles intersect at most twice and also a line intersects any unit circle at most twice, the zone has complexity $O(\tau 2^{\alpha(\tau)}) \subset O(\tau^2)$, where $\alpha(\cdot)$ denotes the inverse Ackermann function [39, Theorem 5.11]. Thus we can compute all lists in $O(\tau^{12})$ time.

Given the list $L_{\phi(s)}$, the partial signature of s is determined by the position of the endpoints of s in $L_{\phi(s)}$. There are $O(\tau^2)$ possible ways for this, and we build a table $T_{\phi(s)}$ that represents them. For each entry in $T_{\phi(s)}$, we store an identifier for the corresponding partial signature.

The total size of the data structure is $O(\tau^{12})$ and it can be constructed in the same time. A query takes $O(\log \tau)$ steps: given s, we can compute ℓ_s^*, ℓ_{sa}^* and ℓ_{sb}^* in constant time. Then we perform repeated binary searches in $\mathcal{S}$ to determine $\phi(s)$. With an appropriate tree structure, we can then find the list $L_{\phi(s)}$ in $O(\log \tau)$ steps. The locations of the endpoints of s in $L_{\phi(s)}$ are found with two more binary searches, and they are then used to look up the partial signature for s in the table $T_{\phi(s)}$, again using binary search. □

Using the data structure from the above lemma, we can determine the partial signature for each row in each vertical τ-strip in total time proportional to

$$\frac{n}{\tau}(\tau^c + n \log \tau) = n\tau^{c-1} + \frac{n^2 \log \tau}{\tau}.$$

We repeat the procedure with the horizontal strips. Now we know for each elementary box in $\mathrm{FSD}(P, Q)$ the partial signature for each row and each column. Next we would like to put them together to find the corresponding entry in our large lookup table. There are several ways to do

this. One way uses table lookup and requires the word RAM, the other way works on the pointer machine, but is a bit more involved.

Table Lookup. We organize the big lookup table from the previous section as a large tree. Each level of the tree corresponds to a row or column of the elementary box, so there are 2τ levels. Each node has $(2\tau)!$ children, representing the possible signatures for the next row or column. We will choose τ so small that a signature can be represented by a single word. Thus, each node of the tree can store an array for its children, and we can choose the appropriate child for a given elementary box in constant time. Thus, to determine the partial signature for each elementary box requires $O(\tau)$ steps in a word RAM, for a total of $O(n^2/\tau)$.

Pointer Juggling. On a pointer machine, we are not allowed to store a lookup table on every level of the tree. Instead, we have a record for each of the $(2\tau)!$ permutations of the set Z. The signature of a row or column of an elementary box is represented by a pointer to the corresponding record. We propagate the elementary boxes through the tree simultaneously, level by level. In the first level, all elementary boxes are assigned to the root. Then, we go through the nodes of one level of the tree, from left to right. We consider each elementary box assigned to the current node, and put it in the bucket for the appropriate signature—the bucket is addressed through the record that represents the permutation, so we use the same buckets for all nodes of the tree. Then we go through the nodes of the next level. The boxes assigned to a node in the tree occur consecutively in the corresponding bucket, and can be popped for the node and stored there. The total time for this procedure is $O(n^2/\tau^2)$ for each level, plus the number of nodes in the tree. We choose τ such that the tree has $o(n)$ nodes, so the total time is again $O(n^2/\tau)$.

As a consequence we obtain the following lemma.

Lemma 4.2. *The partial signature for each elementary box can be determined in time* $O(n\tau^{c-1} + n^2/\tau)$. □

5 Processing the Free Space Diagram

Lemma 5.1. *If the partial signature for each elementary box is known, we can determine whether* $(n, n) \in \text{reach}(P, Q)$ *in time* $O(n^2(\log \tau)/\tau)$.

Proof. We go through the elementary boxes of $\text{FSD}(P, Q)$, processing them one column at a time, from bottom to top. Initially, for the box S in the lower left corner of $\text{FSD}(P, Q)$, we know the whole signature for S, and we can use the signature to determine the intersections of $\text{reach}(P, Q)$ with the upper and right boundary of S (the signature gives us the combinatorial structure, from which we can find the actual intervals in total time $O(\tau)$). Given that partial signatures of the adjacent boxes, we can determine the full signature in total time $O(\tau \log \tau)$: we already have a partial permutation for each row and column of the elementary box, and we only need to determine the position of s_1, t_1 in this permutation. Given the actual values, we can figure this out in $O(\log \tau)$ steps using binary search. Thus, the complete signatures of the adjacent boxes can be determined in total time $O(\tau \log \tau)$ per box, so the total time is $O(n^2(\log \tau)/\tau)$, as claimed. □

Theorem 5.2. *The decision version of the Fréchet problem can be solved in time*

$$O(n^2(\log \log n)^{3/2}/\sqrt{\log n})$$

on a pointer machine.

Proof. Set $\tau := (1/100)\sqrt{\log n / \log\log n}$, then $\tau^{O(\tau^2)} = o(n)$, so we can apply the above lemmas in sequence. □

6 An Improved Bound Using the Word RAM

We now sketch how the running time of our algorithm can be improved if our computational model allows for constant time table-lookup. We will use the same τ as above.

We introduce a second level in the free-space diagram: a *cluster* is a collection of $\tau \times \tau$ elementary boxes. Thus, a cluster corresponds to $\tau^2 \times \tau^2$ boxes in $\mathrm{FSD}(P, Q)$. Fix a cluster C and let R be a row of C. Similarly to before, the row R corresponds to a line segment e on Q and a subcurve P' of P with τ^2 elements. We associate with R an ordered set $Z = \langle z_1, z'_1, z_2, z'_2, \ldots, 2z_{\tau^2}, 2z'_{\tau^2}\rangle$ of $4\tau^2$ elements. The z_i elements represent the order in which the unit circles around the vertices of P' intersect the line segment e, and the z'_i elements are supposed to represent the positions of the reach-intervals on the left boundary. where circles from before P' can intersect e. The *extended signature* for a row R' of an elementary box inside R consists of the subset of Z that corresponds to the subsubcurve P'' of P' for R' plus a permutation that represents the order of the doors inside R', as before. Since there are $O(\tau^{2\tau})$ relevant subsets of size τ, the number of signatures for R' is still $\tau^{O(\tau)}$. We define the signatures for the columns analogously, and we concatenate the signatures to obtain the signature for a whole elementary box. Hence, the total number of possible signatures remains $\tau^{O(\tau^2)}$.

Now for each extended signature we build a lookup table as follows: the input is a word that consists of 4τ fields that store the indices in Z of the incoming doors for the elementary box (2τ fields for the left boundary and 2τ fields for the lower boundary). The output consists of a word that represents the indices for the elements in Z that represent the output doors for the upper and right boundary of the box. The input of the table consists of $O(\tau \log \tau)$ bits, so the size of the table is $\tau^{O(\tau)}$.

During the preprocessing for a given input P, Q, we increase the size of a strip to τ^2, where each strip now consists of τ *substrips* of τ elements. Lemma 4.1 still holds, with a larger constant c. The data structure yields for a query segment a word that contains τ fields of length $O(\tau \log \tau)$, each containing the partial signature for the corresponding row in an elementary box. The total time for building all the data structures and processing all rows is thus $O(n/\tau^2\ (\tau^c + n \log \tau)) = O(n^2(\log \tau)/\tau^2)$. Now we need to put this information together in order to obtain the partial signatures for the vertical sides of the elementary boxes. For this, we need the following lemma, which can be found in Thorup [40].

Lemma 6.1. *Let X be a sequence of τ words that contain τ fields each, so that X can be interpreted as a $\tau \times \tau$ matrix. Then we can compute in time $O(\tau \log \tau)$ on a word RAM a sequence Y of τ words with τ fields each that represents the* transposition *of X.*

Proof (sketch). The algorithm uses a simple divide and conquer approach. The recursive algorithm solves a more general problem: let X be a sequence of a words that represents a sequence M of b $a \times a$ matrices, such that the i-th word in X contains the fields of the i-th row of each matrix in M from left to right. Compute a sequence of words Y that represents the sequence M' of the transposed matrices in M.

The recursion works as follows: if $a = 1$, there is nothing to be done. Otherwise, we split X into the sequence X_1 of the first $a/2$ words and the sequence X_2 of the remaining words. X_1 and

X_2 now represent a sequence of $2b$ $(a/2) \times (a/2)$ matrices, which we transpose recursively. After the recursion, we put the $(a/2) \times (a/2)$ submatrices back together in the obvious way. To finish, we need to transpose the off-diagonal submatrices. This can be done simultaneously for all matrices in time $O(a)$, by using appropriate bit-operations (or table lookup).

Hence, the running time obeys a recursion of the form $T(a) = 2T(a/2) + O(a)$, which solves to $T(a) = O(a \log a)$, as desired. □

By applying the lemma to the words that represent τ consecutive rows in a strip, we can obtain the vertical signature for each elementary box. This takes total time $O((n/\tau^2) \cdot (n/\tau) \cdot \tau \log \tau) = O(n^2(\log \tau)/\tau^2)$. By repeating for the horizontal strips, and using an appropriate lookup table to combine the vertical and horizontal partial signatures, we can thus obtain the signature for each elementary box in total time $O(n^2(\log \tau)/\tau^2)$.

Finally, we describe how to perform the actual computation. We traverse the free-space diagram cluster by cluster (recall that a cluster consists of $\tau \times \tau$ elementary boxes). The clusters are processed column by column from left to right, and inside each column from the bottom to the top. Before processing a cluster, we walk along the left and lower boundary of the cluster to determine the incoming doors. This is done by performing a binary search for each box on the boundary, and determining the appropriate elements z'_i which correspond to the incoming doors. Using this information, we can assemble to appropriate words that represent the incoming information for each elementary box. Since there are n^2/τ^4 clusters, this step requires time $O((n^2/\tau^4)\tau^2 \log \tau) = O(n^2(\log \tau)/\tau^2)$. We then process the elementary boxes inside the cluster, in a similar fashion. Now, however, we can process each elementary box in constant time through a single table lookup, so the total time is $O(n^2/\tau^2)$. Hence, the total running time of our algorithm is $O(n^2(\log \tau)/(\tau^2)) = O(n^2(\log \log n)^2/\log n)$, by our choice of $\tau = \Theta(\sqrt{\log n/\log \log n})$.

7 Decision Trees

Using the techniques from Agarwal et al. we can obtain the following theorem[3].

Theorem 7.1. *There exists an algebraic computation tree for the* discrete *Fréchet problem of depth* $\widetilde{O}(n^{3/2})$.

Proof. We first discuss the decision problem. For the discrete case, the analogue of the free-space diagram is just a $n \times n$ boolean matrix M, where the bit in the i-th row and the j-th column indicates whether the pair (p_i, q_j) can be reached from (p_0, q_0) through a sequence of legal Fréchet moves.

We set $\tau = \sqrt{n}$ and subdivide the columns of M into strips of width τ, as above. Each strip corresponds to a subsequence Q' of τ points q_j, and we compute the arrangement of unit disks with centers in Q'. This takes time $O(\tau^2)$. Then we locate each point of P in this arrangement, taking time $O(n \log n)$. We do this for every strip, resulting in a total running time of $\widetilde{O}(n^{3/2})$, by our choice of τ. As observed by Agarwal et al., the information we gain in this way suffices to complete M without further comparisons on the input. Using the techniques from Agarwal et al., one can then solve the optimization problem while losing only another log-factor, which is absorbed into the $\widetilde{O}$-notation. □

We can prove an analogous statement for the continuous Fréchet distance.

[3] The notation $\widetilde{O}(\cdot)$ stands for $O(\cdot)$ up to logarithmic factors.

Theorem 7.2. *There exists an algebraic decision tree for the Fréchet problem (decision version) of depth $O(n^{2-\varepsilon})$.*

Proof. The reason of our choice for τ was to keep the time for the first preprocessing polynomial. If we only care about the depth of the associated decision tree, we can disregard the cost for building the lookup tables, because those do not depend on the input. Thus, choosing $\tau = n^{1/12}$, we get a depth of the decision tree of (assuming that $c = 12$)

$$\frac{n}{n^{1/12}}n + \frac{n^2 \log n}{n^{1/12}} = O(n^{23/12} \log n) = O(n^{2-\varepsilon}).$$

□

This has the following consequence for Alt's conjecture (assuming linear time reductions):

Corollary 7.3. *If the decision version of the Fréchet problem is* 3SUM*-hard, then* 3SUM *has an algebraic decision tree of depth $O(n^{2-\varepsilon})$.*

We leave it to the reader to judge the implications on the status of the conjecture.

8 Solving the Optimization Problem

The optimization version of the Fréchet problem, that is, computing the Fréchet distance, can be done in $O(n^2 \log n)$ time using parametric search with the decision version as a subroutine. We showed that the decision problem can be solved in $o(n^2)$ time. This however does not directly yield a faster algorithm for the optimization problem: If the running time of the decision problem is $T(n)$ steps, parametric search results in an $O((T(n) + n^2) \log n)$ time algorithm [6]. There is an alternative randomized algorithm by Raichel and Har-Peled [35], which however also runs in $O((T(n) + n^2) \log n)$ time. We will adapt this algorithm to speed up the optimization problem.

Before we do so, we recall that possible values of the Fréchet distance can be limited to a certain set of *critical values* [6]:

1. The distance between a vertex of the one curve and a vertex of the other curve (**vertex-vertex**),
2. The distance between a vertex of the one curve and an edge of the other curve (**vertex-edge**),
3. For two vertices of one curve and an edge of the other curve the distance between one of the vertices and the intersection of e with the bisector of the two vertices (if this intersection exists) (**vertex-vertex-edge**).

If we also include vertex-vertex-edge tuples with no intersection, we can sample a critical value uniformly at random in constant time. The algorithm now works as follows (see Har-Peled and Raichel [35] for more details): it first samples $K = 4n^2$ critical values uniformly at random. Next the algorithm finds the interval $[a, b]$ with a and b being two critical values in the sample such that the Fréchet distance lies in $[a, b]$, and no other critical value of the sample lies in $[a, b]$. In the original algorithm this is done by sorting the critical values and performing a binary search using the decision version. By using median-finding instead, this step can be done in $O(K + T(n) \log K)$ time. We note that the running time of this step could alternatively be reduced by picking a smaller K.

Now, this atomic interval $[a, b]$ of the sampled critical values with high probability only contains a small number of the remaining critical values. More specifically, for $K = 4n^2$ the probability that the interval contains more than $2cn \ln n$ critical values in bounded from above by $1/n^c$ [35, Lemma 6.2].

The remainder of the algorithm first determines these critical values, then again sorts them and performs a binary search. We note that with median-finding the last step, that is, excluding the time to determine the critical values, takes $O(K' + T(n) \log K')$ time. Thus the crucial part is to determine the K' critical values fast.

In $O(n^2)$ time we can check for any vertex-vertex and vertex-edge pair whether the corresponding critical value lies in $[a, b]$. It remains to determine the critical values corresponding to vertex-vertex-edge tuples. These critical values are determined by a variant of the standard sweepline algorithm. For this take an edge e of P and the vertices of Q. The sweep starts with circles of radius a around the vertices of Q and increases the radii until they reach b. During this sweep the algorithm maintains the order in which the circle arcs intersect e. A critical value of the vertex-vertex-edge type corresponds to the event that two different circles intersect e in the same point. Next to these events the sweepline algorithm requires the following events: a circle intersects e for the first time, or a circle intersects one of the vertices of e. Both of these event types correspond to critical values involving e or a vertex of e. Thus if we perform such a sweep for all edges of P (and similarly for the edges of Q), the total number of events will be $O(K')$, thus the overall running time of all sweeps ignoring the time for initialization is $O(K' \log n)$.

It remains to show that we can compute the initial order in which the circle arcs intersect e fast. First compute the arrangement $\mathcal{A}$ of circles with radius a around the vertices of Q. This can be done in $O(n^2)$ time [23]. We need to determine in which order the arcs of the circles intersect e. We can determine the order of intersections by traversing in $\mathcal{A}$ the zone of the line ℓ spanned by e. The time for the traversal can be bounded by the complexity of the zone. Using that the circles pairwise intersect at most twice and the line intersects each circle also only twice, the complexity of the zone can be bounded by $O(n2^{\alpha(n)})$ [39, Theorem 5.11]. Summing over all edges e this adds a total of $O(n^2 2^{\alpha(n)})$ to the running time. Thus the overall running time is $O(T(n) \log(n) + n^2 2^{\alpha(n)} + K' \log n)$. The case that $K' > 8n \ln n$ happens with probability less than $1/n^4$, and also in this case K' is still in $O(n^3)$. Thus, this case adds $o(1)$ to the expected running time. The case $K' \leq 8n \ln n$ adds $O(n \log^2 n)$ to the expected running time. As a consequence we obtain the following lemma.

Lemma 8.1. *The Fréchet distance of two polygonal curves with n vertices each can be computed by a randomized algorithm in $O(n^2 2^{\alpha(n)} + T(n) \log n)$ expected time, where $T(n)$ is the running time for the decision problem*

We plug in our new bound on $T(n)$.

Theorem 8.2. *The Fréchet distance of two polygonal curves with n vertices each can be computed by a randomized algorithm in time $O(n^2 \sqrt{\log n} (\log \log n)^{3/2})$ on a pointer machine and in time $O(n^2 (\log \log n)^2)$ on a word RAM.*

9 Conclusion

In this paper we break the long-standing quadratic upper bound for the decision version of the Fréchet problem. Moreover, we show that this problem has an algebraic decision tree of depth

$O(n^{2-\varepsilon})$, for some $\varepsilon > 0$ and where n is the number of vertices of the polygonal curves. This strongly indicates that the problem is not 3SUM-hard after all. We show how our faster algorithm for the decision version can be used for a faster algorithm to compute the Fréchet distance. If we allow constant-time table-lookup, we obtain a running time in close reach of $O(n^2)$.

This leaves us with intriguing open research questions. Can we reduce the time needed for the decision version to $O(n^{2-\varepsilon})$, that is the bound we obtain from the algebraic decision tree? Can we devise a quadratic or even subquadratic algorithm for the optimization version? Can we devise such an algorithm on the word RAM, that is, with constant-time table-lookup? Or, on the other hand, can we establish a connection between the Fréchet distance and other problems which exhibit a discrepancy between the decision tree and the uniform complexity, such as, e.g., Min-Plus-Convolution?

Acknowledgments

We would like to thank Natan Rubin for pointing out to us that Fréchet-related papers require a witty title involving a dog. We would like to thank Bettina Speckmann for inspiring discussions during the early stages of this research.

References

[1] P. K. Agarwal, R. B. Avraham, H. Kaplan, and M. Sharir. Computing the discrete Fréchet distance in subquadratic time. To appear in *Proc. Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, 2013. `arXiv:1204.5333`.

[2] P. K. Agarwal, S. Har-Peled, N. H. Mustafa, and Y. Wang. Near-linear time approximation algorithms for curve simplification. *Algorithmica*, 42(3–4):203–219, 2005.

[3] N. Ailon and B. Chazelle. Lower bounds for linear degeneracy testing. *J. ACM*, 52(2):157–171, 2005.

[4] H. Alt. The computational geometry of comparing shapes. In *Efficient Algorithms*, volume 5760 of *Lecture Notes in Computer Science*, pages 235–248. Springer-Verlag, 2009.

[5] H. Alt and M. Buchin. Can we compute the similarity between surfaces? *Discrete Comput. Geom.*, 43(1):78–99, 2010.

[6] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *Internat. J. Comput. Geom. Appl.*, 5(1–2):78–99, 1995.

[7] H. Alt, C. Knauer, and C. Wenk. Comparison of distance measures for planar curves. *Algorithmica*, 38(1):45–58, 2003.

[8] B. Aronov, S. Har-Peled, C. Knauer, Y. Wang, and C. Wenk. Fréchet Distances for Curves, Revisited. In *Proc. 14th Annu. European Sympos. Algorithms (ESA)*, pages 52–63, 2006.

[9] S. Arora and B. Barak. *Computational complexity. A modern approach*. Cambridge University Press, Cambridge, 2009.

[10] R. Bellman and R. Kalaba. On adaptive control processes. *IRE Transactions on Automatic Control*, 4(2):1–9, 1959.

[11] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On map-matching vehicle tracking data. In *Proc. 31st Int. Conf. on Very Large Data Bases*, pages 853–864. ACM, 2005.

[12] D. Bremner, T. M. Chan, E. D. Demaine, J. Erickson, F. Hurtado, J. Iacono, S. Langerman, and P. Taslakian. Necklaces, convolutions, and $X+Y$. In *Proc. 14th Annu. European Sympos. Algorithms (ESA)*, pages 160–171. Springer-Verlag, 2006.

[13] K. Buchin, M. Buchin, and J. Gudmundsson. Constrained free space diagrams: a tool for trajectory analysis. *Int. J. of GIS*, 24(7):1101–1125, 2010.

[14] K. Buchin, M. Buchin, J. Gudmundsson, M. Löffler, and J. Luo. Detecting commuting patterns by clustering subtrajectories. *Internat. J. Comput. Geom. Appl.*, 21(3):253–282, 2011.

[15] K. Buchin, M. Buchin, C. Knauer, G. Rote, and C. Wenk. How difficult is it to walk the dog? In *Proc. 23rd European Workshop Comput. Geom. (EWCG)*, pages 170–173, 2007.

[16] K. Buchin, M. Buchin, W. Meulemans, and B. Speckmann. Locally correct Fréchet matchings. In *Proc. 20th Annu. European Sympos. Algorithms (ESA)*, 2012.

[17] K. Buchin, M. Buchin, and A. Schulz. Fréchet distance of surfaces: Some simple hard cases. In *Proc. 18th Annu. European Sympos. Algorithms (ESA)*, pages 63–74, 2010.

[18] K. Buchin, M. Buchin, and Y. Wang. Exact algorithms for partial curve matching via the Fréchet distance. In *Proc. 20th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 645–654, 2009.

[19] K. Buchin, M. Buchin, and C. Wenk. Computing the Fréchet distance between simple polygons. *Comput. Geom. Theory Appl.*, 41(1–2):2–20, 2008.

[20] K. Buchin and W. Mulzer. Delaunay triangulations in $O(\mathrm{sort}(n))$ time and more. *J. ACM*, 58(2):Art. 6, 27, 2011.

[21] M. Buchin. *On the Computability of the Fréchet Distance Between Triangulated Surfaces*. PhD thesis, Free University Berlin, Institute of Computer Science, 2007.

[22] E. Chambers, É. de Verdière, J. Erickson, S. Lazard, F. Lazarus, and S. Thite. Homotopic Fréchet distance between curves or, walking your dog in the woods in polynomial time. *Comput. Geom. Theory Appl.*, 43(3):295–311, 2010.

[23] B. M. Chazelle and D. T. Lee. On a circle placement problem. *Computing*, 36(1–2):1–16, 1986.

[24] A. F. Cook, A. Driemel, S. Har-Peled, J. Sherette, and C. Wenk. Computing the Fréchet distance between folded polygons. In *Proc. 12th International Symposium on Algorithms and Data Structures (WADS)*, pages 267–278, 2011.

[25] A. F. Cook and C. Wenk. Geodesic Fréchet distance inside a simple polygon. *ACM Transactions on Algorithms*, 7(1):9, 2010.

[26] M. de Berg, A. F. Cook IV, and J. Gudmundsson. Fast Fréchet queries. In *Proc. 22nd Annu. Internat. Sympos. Algorithms Comput. (ISAAC)*, pages 240–249, 2011.

[27] D. Dobkin and R. J. Lipton. Multidimensional searching problems. *SIAM J. Comput.*, 5(2):181–186, 1976.

[28] A. Driemel and S. Har-Peled. Jaywalking your dog: computing the Fréchet distance with shortcuts. In *Proc. 23rd Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 318–337, 2012.

[29] A. Driemel, S. Har-Peled, and C. Wenk. Approximating the Fréchet distance for realistic curves in near linear time. In *Proc. 26th Annu. ACM Sympos. Comput. Geom. (SoCG)*, pages 365–374. ACM, 2010.

[30] M. L. Fredman. How good is the information theory bound in sorting? *Theoret. Comput. Sci.*, 1(4):355–361, 1975/76.

[31] A. Gajentaan and M. H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom. Theory Appl.*, 5(3):165–185, 1995.

[32] M. Godau. *On the complexity of measuring the similarity between geometric objects in higher dimensions*. PhD thesis, Freie Universität Berlin, Germany, 1998.

[33] J. Gudmundsson and T. Wolle. Towards automated football analysis: Algorithms and data structures. In *Proc. 10th Australasian Conf. on Mathematics and Computers in Sport*, 2010.

[34] S. Har-Peled, A. Nayyeri, M. Salavatipour, and A. Sidiropoulos. How to walk your dog in the mountains with no magic leash. In *Proc. 28th Annu. ACM Sympos. Comput. Geom. (SoCG)*, pages 121–130. ACM, 2012.

[35] S. Har-Peled and B. Raichel. The Fréchet distance revisited and extended. In *Proc. 27th Annu. ACM Sympos. Comput. Geom. (SoCG)*, pages 448–457. ACM, 2011.

[36] P. Indyk. Approximate nearest neighbor algorithms for Frechet distance via product metrics. In *Proc. 18th Annu. ACM Sympos. Comput. Geom. (SoCG)*, pages 102–106. ACM, 2002.

[37] A. Maheshwari, J.-R. Sack, K. Shahbaz, and H. Zarrabi-Zadeh. Fréchet distance with speed limits. *Comput. Geom. Theory Appl.*, 44(2):110–120, 2011.

[38] A. Maheshwari, J.-R. Sack, K. Shahbaz, and H. Zarrabi-Zadeh. Improved algorithms for partial curve matching. In *Proc. 19th Annu. European Sympos. Algorithms (ESA)*, pages 518–529, 2011.

[39] M. Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, 1995.

[40] M. Thorup. Randomized sorting in $O(n \log \log n)$ time and linear space using addition, shift, and bit-wise boolean operations. *J. Algorithms*, 42(2):205–230, 2002.

[41] C. Wenk, R. Salas, and D. Pfoser. Addressing the need for map-matching speed: Localizing global curve-matching algorithms. In *Proc. 18th Int. Conf. on Sci. and Stat. Database Management*, pages 379–388, 2006.