

Computing the Discrete Fréchet Distance in Subquadratic Time*

Pankaj K. Agarwal[†] Rinat Ben Avraham[‡] Haim Kaplan[§] Micha Sharir[¶]

November 5, 2018

Abstract

The Fréchet distance is a similarity measure between two curves A and B : Informally, it is the minimum length of a leash required to connect a dog, constrained to be on A , and its owner, constrained to be on B , as they walk without backtracking along their respective curves from one endpoint to the other. The advantage of this measure on other measures such as the Hausdorff distance is that it takes into account the ordering of the points along the curves.

The discrete Fréchet distance replaces the dog and its owner by a pair of frogs that can only reside on n and m specific pebbles on the curves A and B , respectively. These frogs hop from a pebble to the next without backtracking. The discrete Fréchet distance can be computed by a rather straightforward quadratic dynamic programming algorithm. However, despite a considerable amount of work on this problem and its variations, there is no subquadratic algorithm known, even for approximation versions of the problem.

In this paper we present a subquadratic algorithm for computing the discrete Fréchet distance between two sequences of points in the plane, of respective lengths $m \leq n$. The algorithm runs in $O\left(\frac{mn \log \log n}{\log n}\right)$ time and uses $O(n + m)$ storage. Our approach uses the geometry of the problem in a subtle way to encode legal positions of the frogs as states of a finite automata.

*Work on this paper by Pankaj Agarwal and Micha Sharir has been supported by Grant 2006/194 from the U.S.–Israel Binational Science Foundation. Work by Pankaj Agarwal is also supported by NSF under grants CNS-05-40347, CCF-06 -35000, IIS-07-13498, and CCF-09-40671, by ARO grants W911NF-07-1-0376 and W911NF-08-1-0452, by an NIH grant 1P50-GM-08183-01, and by a DOE grant OEG-P200A070505. Work by Haim Kaplan has been supported by Grant 2006/204 from the U.S.–Israel Binational Science Foundation, and by Grant 822/10 from the Israel Science Fund. Work by Micha Sharir has also been supported by NSF Grant CCF-08-30272, by Grant 338/09 from the Israel Science Fund, and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University. Work by Haim Kaplan and Micha Sharir has also been supported by the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11). Work by Rinat Ben Avraham was supported by the Israel Science Fund Grants 338/09 and 822/10.

[†]Department of Computer Science, Box 90129, Duke University, Durham, NC 27708-0129, USA; pankaj@cs.duke.edu

[‡]School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel; rinatba@gmail.com

[§]School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel; haimk@post.tau.ac.il

[¶]School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel; and Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, USA; michas@post.tau.ac.il

1 Introduction

Problem statement. Let $A = (a_1, \dots, a_m)$ and $B = (b_1, \dots, b_n)$ be two sequences of m and n points, respectively, in the plane. The *discrete Fréchet distance* $\delta_{dF}(A, B)$ between A and B is defined as follows. Fix a distance $\delta > 0$ and consider the Cartesian product $A \times B$ as the vertex set of a directed graph G_δ whose edge set is

$$E_\delta = \left\{ \left((a_i, b_j), (a_{i+1}, b_j) \right) \mid \|a_i - b_j\|, \|a_{i+1} - b_j\| \leq \delta \right\} \cup \left\{ \left((a_i, b_j), (a_i, b_{j+1}) \right) \mid \|a_i - b_j\|, \|a_i - b_{j+1}\| \leq \delta \right\};$$

here we consider the case where $\|\cdot\|$ is the Euclidean norm. Then $\delta_{dF}(A, B)$ is the smallest $\delta > 0$ for which (a_m, b_n) is reachable from (a_1, b_1) in G_δ . Informally, think of A and B as two sequences of stepping stones, and of two frogs, the A -frog and the B -frog, where the A -frog has to visit all the A -stones in order, and the B -frog has to visit all the B -stones in order. The frogs are connected to each other by a rope of length δ , and are initially placed at a_1 and b_1 , respectively. At each move, exactly one of the frogs can jump from its current stone to the next one, which can be done if and only if its distances to the other frog, before and after the jump, are both at most δ (see Figure 2 for an example of a possible sequence of jumps of the two frogs). Then $\delta_{dF}(A, B)$ is the smallest $\delta > 0$ for which there exists a sequence of jumps that gets the frogs to a_m and b_n , respectively. (Note that the frogs cannot backtrack.)

Remark. In this formulation we forbid the frogs to jump simultaneously, from a placement (a_i, b_j) to (a_{i+1}, b_{j+1}) . However, our algorithm can be modified so that it also applies to the variant where such “diagonal” moves are also allowed. (See a remark in Section 2.1.)

The continuous Fréchet distance. The discrete Fréchet distance problem is a variant of the (more standard, continuous) Fréchet distance problem. Informally, consider a person and a dog connected by a leash, each walking along a path (curve) from its starting point to its end point. Both are allowed to control their speed, but they cannot backtrack. The Fréchet distance between the two curves is the minimal length of a leash that is sufficient for traversing both curves in this manner.

More formally, a curve $f \subseteq \mathbb{R}^2$ is a continuous mapping from $[0, 1]$ to \mathbb{R}^2 . A *reparameterization* is a continuous nondecreasing surjection $\alpha : [0, 1] \rightarrow [0, 1]$, such that $\alpha(0) = 0$ and $\alpha(1) = 1$. The Fréchet distance $\delta_F(f, g)$ between two curves f and g is then defined as follows:

$$\delta_F(f, g) = \inf_{\alpha, \beta} \max_{t \in [0, 1]} \left\{ \|f(\alpha(t)) - g(\beta(t))\| \right\},$$

where $\|\cdot\|$ is the underlying norm (typically, the Euclidean norm), and α and β are reparameterizations of $[0, 1]$.

The semi-continuous Fréchet distance. One may also consider a hybrid version of the problem, of a person walking a frog. Formally, we have a curve f and a sequence B of n stepping stones. We want to find the smallest $\delta > 0$ for which f can be partitioned into n

(pairwise openly disjoint) arcs f_1, \dots, f_n so that the distance of b_i from every point of f_i is at most δ , for $i = 1, \dots, n$. In this setup, for each $i = 1, \dots, n$, the person walks along f_i from its starting point to its end point, while the frog sits at b_i . Then, when the person reaches the endpoint, the frog jumps to b_{i+1} , and they keep moving this way until all of f and B are traversed.

Remark. All three variants of the Fréchet distance can be extended, in an obvious manner, to any dimension $d \geq 2$, but in this paper we only consider the planar case.

Background. Motivated by a variety of applications, the Fréchet distance has been studied extensively in computational geometry for the past 20 years, as a useful measure for the similarity between curves [7, 9]. If data is uniformly sampled, which is often the case in practice, it suffices to compute the discrete Fréchet distance between the sequences of vertices of the two curves. The extended model that also allows diagonal moves (as in a preceding remark) can potentially allow us to sample more sparsely along relatively straight portions of the curves.

Eiter and Mannila [14] showed that the discrete Fréchet distance in the plane can be computed in quadratic time (that is, in $O(mn)$ time). Later, Aronov et al. [5] have given a $(1 + \varepsilon)$ -approximation algorithm which solves the discrete Fréchet distance problem between the vertices of two *backbone curves* in near linear time. Backbone curves are required to have edges whose lengths are close to 1, and a constant lower bound on the minimal distance between any pair of vertices; they model, e.g., the backbone chains of proteins. Concerning the continuous Fréchet distance problem, Alt and Godau [3] have shown that the Fréchet distance of two polygonal curves with a total of n edges in the plane can be computed in $O(n^2 \log n)$ time. A lower bound of $\Omega(n \log n)$ time for the decision version of the problem, where the task is to decide whether the Fréchet distance between two curves is smaller than or equal to a given value, was given by Buchin et al. [8]. They also showed that this bound holds for the discrete version of the problem as well. It has been an open problem to compute (exactly) the continuous or discrete Fréchet distance in subquadratic time. Even the simpler variant, in which we only want to solve the decision version of the discrete Fréchet distance problem in the plane in subquadratic time has still been open. In fact, only a few years ago, Alt [2] has conjectured that the decision subproblem of the (continuous) Fréchet distance problem is 3SUM-hard [15].

We note that it is also an open problem to solve the approximation versions of the Fréchet distance problems in subquadratic time. That is, no subquadratic algorithm (in m and n , with any reasonable dependence on ε) is known for computing a $(1 + \varepsilon)$ -approximation of either variant of the Fréchet distance (for arbitrary curves / sequences, with no restrictions on their shape).

To date, the only subquadratic algorithms known for the Fréchet distance problem (either continuous or discrete) are for restricted classes of curves, such as the algorithm of Aronov et al. [5] mentioned above. Other classes of curves considered so far in the literature include closed convex curves and κ -bounded curves [4]. A curve is κ -bounded if, for any pair of points a, b on the curve, the portion of the curve between a and b is contained in $D(a, \frac{\kappa}{2}\|a - b\|) \cup D(b, \frac{\kappa}{2}\|a - b\|)$, where $D(p, r)$ denotes the disk of radius r centered at p . Alt et al. [4] showed that the Fréchet distance between two convex curves equals their

Hausdorff distance, and that the Fréchet distance between two κ -bounded curves is at most $(1 + \kappa)$ times their Hausdorff distance, and thus an $O(n \log n)$ algorithm for computing or approximating the Hausdorff distance (as given in [2]) can be applied to obtain an efficient exact solution in the convex case or a constant-factor approximation in the κ -bounded case. Later, Driemel et al. [13] provided a $(1 + \varepsilon)$ -approximation algorithm for *c-packed curves* in \mathbb{R}^d that runs in $O(cn/\varepsilon + cn \log n)$ time, where a curve π is called *c-packed* if the total length of π inside any ball is bounded by c times the radius of the ball.

Another variant of the Fréchet distance is the *weak* Fréchet distance, which, in the person-dog scenario, allows the person and the dog to also walk backwards. Recently, Har-Peled and Raichel [16] gave a quadratic algorithm for computing (a generalization of) the weak Fréchet distance between curves. More specifically, given two simplicial complexes in \mathbb{R}^d , and start and end vertices in each complex, they show how to compute two curves in these complexes that connect the corresponding start and end points, such that the weak Fréchet distance between these curves is minimized. Since a polygonal curve is a simplicial complex, this can be viewed as a generalization of the regular notion of the weak Fréchet distance between curves.

See also [10, 12] for a few additional results on the Fréchet distance.

Our results. We present a new algorithm for computing the discrete Fréchet distance whose running time is $O(mn \log \log n / \log n)$ (assuming $m \leq n$). We first present a procedure for solving the decision version of the problem: Given $\delta > 0$, determine whether the discrete Fréchet distance between A and B is $\leq \delta$. The decision procedure runs in $O(mn \log \log n / \log^2 n)$ time and uses $O(m + n)$ space. To obtain a solution for the optimization problem, we combine the decision procedure with a relatively simple explicit binary search, based on a simple procedure for distance selection [1]. This increases the total running time by only a factor of $O(\log n)$, so the overall algorithm runs in $O(mn \log \log n / \log n)$ time, which is still subquadratic. Using (a variant of) the procedure in [1], the space required by the optimization algorithm remains linear in $m + n$. The following presentation is therefore mainly focused on the decision procedure, which is the more involved part of our algorithm.

Although not detailed in this abstract, our technique can be extended so as to compute, within the same time bound, (i) the discrete Fréchet distance between two sequences of points in \mathbb{R}^d , for any $d \geq 3$, and (ii) the semi-continuous Fréchet distance between a sequence of points and a curve in the plane. (We do not have at the moment a similar extension to the continuous Fréchet distance, which is one of the main open problems raised by our work.)

A brief sketch of the decision procedure. Let us first provide a brief description of the decision procedure for a given $\delta > 0$. We begin by presenting a slightly less efficient but considerably simpler solution, on which we will then build our improved solution. Consider the following 0/1 matrix M , whose rows (resp., columns) correspond to the points of A (resp., of B). An entry $M_{i,j}$ of M is equal to 1 if the pair (a_i, b_j) is reachable from the starting placement (a_1, b_1) of the trip with a “leash” of length δ . Otherwise, $M_{i,j}$ is equal to 0. In other words, $M_{i,j} = 1$ if the discrete Fréchet distance δ_{dF} between the two prefix subsequences (a_1, \dots, a_i) and (b_1, \dots, b_j) is at most δ , and $M_{i,j} = 0$ otherwise. Thus

determining the value of $M_{m,n}$ solves the overall decision problem.

$M_{m,n}$ can be obtained by computing all entries of M using dynamic programming, as follows. If $\|a_1 - b_1\| \leq \delta$, we set $M_{1,1} := 1$; otherwise, $M_{1,1} := 0$ and the decision procedure is aborted right away, since δ is too small even for the initial placement. The other elements of the first row of M are then filled in order. Specifically, for each $1 < j \leq n$ we set $M_{1,j} := 1$ if (a) $M_{1,j-1} = 1$, and (b) $\|a_1 - b_j\| \leq \delta$; otherwise we set $M_{1,j} := 0$. (Clearly, if $M_{1,j} = 0$, for some $0 \leq j \leq n$, then all the subsequent entries of the first row are also zero.) Similarly, the first column of M is filled in by setting, for each $1 < i \leq m$ in order, $M_{i,1} := 1$ if (a) $M_{i-1,1} = 1$, and (b) $\|a_i - b_1\| \leq \delta$; otherwise, we set $M_{i,1} := 0$. For an arbitrary entry, $1 < i \leq m, 1 < j \leq n$, we set $M_{i,j} := 1$ if (a) at least one of $M_{i,j-1}$ and $M_{i-1,j}$ is 1, and (b) $\|a_i - b_j\| \leq \delta$; otherwise, we set $M_{i,j} := 0$. The cost of this dynamic programming procedure is $O(mn)$.

To obtain a subquadratic decision procedure, we cannot compute each value of M explicitly, and instead we only compute certain rows and columns of M . To be more precise, we partition A into $l = \Theta(m/\log^2 n)$ layers A_1, \dots, A_l , each of length $c_1 \log^2 n$, where $c_1 > 0$ is an appropriate constant such that the last point of any layer A_i is the first point of the next layer A_{i+1} . We can think of this as a partition of M into l “horizontal” strips, each of width $c_1 \log^2 n$, such that the last row of a strip is the first row of the next strip. (See Figure 1 for an illustration.) We then compute, for each strip (in order), the entries of M in the last row of the strip, and we use the values of this row as input for the processing of the next strip.

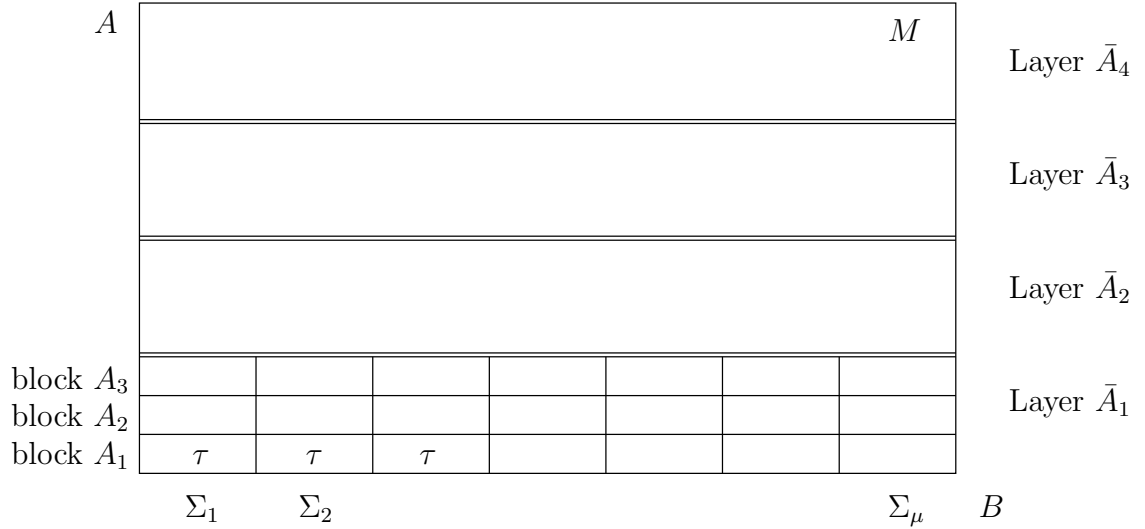


Figure 1: A partition of M into horizontal strips (and substrips), which correspond to layers (and blocks) of A . B is partitioned into subsequences of length τ . Each subsequence of B corresponds to a single symbol Σ_i which the automata \mathcal{K}^* process.

To obtain the running time bound claimed above, we need to compute the entries of M in each of the $l+1$ “boundary” rows (including the first and the last rows) in $O(n \log \log n)$ time. To do so, we further partition each layer A_i into $t = \Theta(\log n)$ blocks, of length $c_2 \log n$ each, where $c_2 > 0$ is a sufficiently small constant to be specified later (or, alternatively, partition each strip of M into $\Theta(\log n)$ substrips, of width $c_2 \log n$ each). As before, the last point

of a block is the first point of the next block. We handle each block in $O(n \log \log n / \log n)$ time, using an approach that resembles the execution of a deterministic finite automaton \mathcal{K}^* . Somewhat informally, the automaton is constructed from the corresponding block of A , and we execute it on a string constructed from the elements of B . To achieve the desired running time (in particular, to avoid having to spend $\Theta(n)$ time in “reading” the individual elements of B), we partition B into $\mu = \Theta(n \log \log n / \log n)$ subsequences of length $\tau = c_3 \log n / \log \log n$ each, where $c_3 > 0$ is yet another constant, and require \mathcal{K}^* to operate on each subsequence, in constant time, as if it were a single symbol.

We note that the compaction of M outlined above is similar to compactations used to solve several related problems. For instance, Baran et al. [6] present an $o(n^2)$ algorithm for the 3SUM problem on integers of bounded length. (See also algorithms for the *edit distance* problem; [11, 17, 19]). However, while the other compactations are purely symbolic, ours is strongly based on the geometry of the problem. A major difference between our algorithm and the other ones is that in our case the input of the problem in itself does not include repetitions (that can be used in the compaction). That is, the input points are not likely to repeat themselves. We create repetitions artificially by constructing the arrangement \mathcal{A} of the disks centered at the points of A , and locating the points of B in this arrangement. Now, the faces of \mathcal{A} that contain the points of B generally repeat themselves. The finite-state automaton \mathcal{K}^* that we construct operates on the faces of \mathcal{A} rather than on the points of B , and this leads to the desired subquadratic performance. Using such an automaton for the compaction appears to be also a novel technique.

Organization. In Section 2 we describe the decision procedure in detail. In particular, in Section 2.1 we show how to deal (slightly less efficiently) with a single block of A . We then show, in Section 2.2 how to handle a layer of A , which contains $\Theta(\log n)$ such blocks, by combining portions of the processing of the separate blocks into a common procedure that is executed at the layer level. Finally, in Section 2.3 we describe the overall decision procedure. (The justification of using blocks of size $\Theta(\log n)$ is deferred to Section 4, where we present a lower-bound construction that indicates that using blocks of larger size may cause our respective automata \mathcal{K}^* to be too large for a subquadratic algorithm.) In Section 3 we show how to combine the decision procedure with an elementary binary search, and obtain the main result of this paper, namely, a subquadratic algorithm for computing the discrete Fréchet distance.

2 The decision procedure

In this section we focus on the decision problem: Given $\delta > 0$, determine whether $\delta_{dF}(A, B) \leq \delta$. By an appropriate scaling, we may assume, without loss of generality, that $\delta = 1$.

As mentioned in the introduction, we partition A into $l = \Theta(m / \log^2 n)$ layers, of size $c_1 \log^2 n$ each (where $c_1 > 0$ is an appropriate constant whose value will be set later), such that the last point of each layer is the first point of the next layer, and process them in order. To process a single layer of A , we further partition it into $t = \Theta(\log n)$ blocks, of size $c_2 \log n$ each (where $c_2 > 0$ is a sufficiently small constant, also to be specified later), such that the last point of each block is the first point of the next block. The algorithm processes the blocks within a layer one by one in order. The purpose of processing a layer is

to collect, in a single processing step, information that will be needed by each of its blocks. Then each block is processed separately, in order, and the M -entries of its terminal row are computed from those in the initial row, all the way to the terminal row of the entire layer.

2.1 Handling a single block of A

Here are the details of processing a single block. To simplify the notation, we denote the block by A ; its size m now satisfies $m = c_2 \log n$ (the very last block of the entire sequence may be smaller). Enumerate the points of A as a_1, \dots, a_m .

Regard the points a_1, \dots, a_m as the centers of respective unit disks D_1, \dots, D_m , and let \mathcal{D} denote the sequence of these disks. Consider the arrangement $\mathcal{A} = \mathcal{A}(\mathcal{D})$ of the disks, and associate with each face f of \mathcal{A} the subset \mathcal{D}_f of disks containing f . For each point $b_i \in B$, denote by f_i the face of \mathcal{A} containing b_i .

Remark. The description given in this subsection provides the essential ingredients of the processing of a block, but is somewhat lax or vague about precise implementation details, which have to be applied with care to ensure the running time we are after. For example, a naive implementation of the step that finds the faces f_i , by n point locations of the points of B in \mathcal{A} , is too expensive for our purpose. The layers are used to conglomerate some parts of the processing within their blocks into a single processing step, thereby improving the efficiency of the procedure. More details are provided in the next subsection.

Fix two indices $1 \leq i \leq j \leq n$, and call the pair $((a_1, b_i), (a_m, b_j))$ *valid* if there exists a path in G_δ (G_1 , that is) from (a_1, b_i) to (a_m, b_j) . We can simulate such a path as a sequence of moves between *basic states*, where each basic state is a pair (f, D_k) , where f is a face of \mathcal{A} and D_k is a disk in \mathcal{D}_f . In each move we either pass from (f, D_k) to (f', D_k) , where f' is another face of \mathcal{A} which is also contained in D_k , or pass from (f, D_k) to (f, D_{k+1}) , if D_{k+1} also belongs to \mathcal{D}_f (i.e., also contains f). See Figure 2. In the original problem (involving the complete unpartitioned A) we would have to start at (f_1, D_1) and to reach (f_n, D_m) (now with m equal to the original size of A), using a sequence of legal moves between basic states, of the types just described, that corresponds to a path in G_1 from (a_1, b_1) to (a_m, b_n) . (For this, though, we would need to construct the huge arrangement of the disks for the entire sequence A , which would have been far too expensive.) In the refined version we start at (f_i, D_1) and have to reach (f_j, D_m) along a similar sequence of moves, for arbitrary indices $i \leq j$ (and for the much smaller size m of a block). This represents the situation where the portion of the trip of the B -frog that corresponds to the passage of the A -frog through the points of the present block A starts at b_i and ends at b_j .

Note that, in view of this interpretation, we are only interested in placements (a_1, b_i) that can be reached (through the preceding blocks of the complete A -sequence) from the starting placement of the whole trip. We refer to such a placement as a *reachable* position of the frogs. Let the flag $\varphi_i = \varphi(b_i)$ indicate whether the placement (a_1, b_i) is reachable through the preceding blocks of A (in which case $\varphi_i = 1$), or not ($\varphi_i = 0$); in this notation we hide the dependence of φ_i on the preceding layers and blocks. Note that if A is the first block, then $\varphi_i = 0$ for each $i > 1$, since (a_1, b_i) is not reachable through the (empty set of) preceding blocks. For $i = 1$ we set $\varphi_1 = 1$ if $b_1 \in D_1$, and otherwise abort the entire procedure, since the frogs lie at their starting positions at distance > 1 .

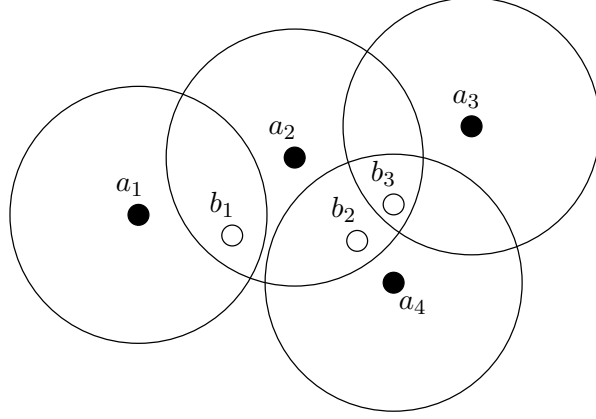


Figure 2: An illustration of the decision problem of the discrete Fréchet distance. The stepping stones of the A -frog are the black points. The disks (of radius δ) centered at the points of A form the arrangement \mathcal{A} . The stepping stones of the B -frog are the hollow points. In this example, a legal path of the two frogs is $((a_1, b_1), (a_2, b_1), (a_2, b_2), (a_2, b_3), (a_3, b_3), (a_4, b_3))$.

We can store the data maintained by this process in more compact form. To do so, we define an *aggregate state* (to which we refer as a *state* for short) to be a pair (f, S_f) , where S_f is a subset of \mathcal{D}_f ; we refer to S_f as the set of valid disks (associated with our state). The set S_f is assumed to have the property, dictated by the transition rules for the frogs, that if $D_k \in S_f$ and $D_{k+1} \in \mathcal{D}_f$ then D_{k+1} also belongs to S_f .

A state (f, S_f) and a pair (g, φ) , where g is a face of \mathcal{A} , and φ is a binary flag, determine a transition into a new state (g, S_g) , where $S_g \subseteq \mathcal{D}_g$ consists of those disks $D_k \in \mathcal{D}_g$ for which there exists $j \leq k$ such that (i) $D_j \in S_f$, and (ii) the entire run D_j, D_{j+1}, \dots, D_k is contained in \mathcal{D}_g . Furthermore, if $\varphi = 1$, then S_g also contains the maximal prefix of disks in \mathcal{D} (starting with D_1) that is contained in \mathcal{D}_g . The passage from (f, S_f) to (g, S_g) is called a *valid transition*.

The interpretation of this setup is as follows. The state (f, S_f) signifies that (a) the B -frog is now at a point that belongs to f , and the A -frog lies at the center a_k of some disk $D_k \in S_f$, and (b) this position has been reached via a legal sequence of interweaving A -moves and B -moves, starting from (a_1, b_1) (if A is the first block of the whole sequence), or from some placement (a_1, b_i) (if A is an intermediate block), which is reachable from the starting positions of the frogs (so φ_i is 1). Moreover, for the specific sequences of stepping stones for the A -frog and the B -frog, the A -frog cannot lie at the center a_k of any disk $D_k \notin S_f$.

The valid transition from (f, S_f) to (g, S_g) means that, for any disk $D_k \in S_g$, we can get the A -frog to lie at its center a_k , and get the B -frog to lie in g , by taking a disk D_j as in the definition of the valid transition, assuming that the A -frog lies at a_j and the B -frog lies in f (in accordance with the above interpretation of (f, S_f)), moving the B -frog to g (which is possible since D_j also belongs to \mathcal{D}_g), and then moving the A -frog through the centers a_{j+1}, \dots, a_k , all at distance at most 1 from the B -frog (or, if $j = k$, let the A -frog stay put). Moreover, if the last move of the B -frog is from f to g , and the A -frog lies at the center of some disk in S_f , then the centers of the disks in S_g are the only possible locations that the A -frog can reach (with this single hop of the B -frog).

In addition, the flag φ allows the B -frog to appear “out of nowhere” in the middle of the first row of the block, in case a position (a_1, b_i) , where $b_i \in g$, is reachable from the starting placement of the whole trip. This means that we can get the A -frog to lie at a_1 , and the B -frog to lie in g , by some path starting at the starting position of the entire trip of the frogs, and moves solely through the points of the preceding blocks of the full sequence A (once the B -frog has reached g , the A -frog can move through the centers of the disks in the prefix of \mathcal{D}_g contained in S_g , and stop at any of these centers before the B -frog makes its next move).

The compression of basic states into aggregate states resembles the construction of a deterministic finite automaton (DFA) from a nondeterministic finite automaton (NFA). This is not accidental; we have already hinted that the algorithm simulates the moves of such an automaton, and the resemblance will become more relevant as we continue to present the algorithm.

Remark: If we want to also consider the variant where the frogs are allowed to jump simultaneously from a placement (a_i, b_j) to (a_{i+1}, b_{j+1}) (provided that $\|a_i - b_j\| \leq 1$ and $\|a_{i+1} - b_{j+1}\| \leq 1$), we only need to modify the above rules of a valid transition. Specifically, a state (f, S_f) and a pair (g, φ) , where g is a face of \mathcal{A} , and φ is a binary flag, determine a transition into a new state (g, S'_g) , where $S'_g \subseteq \mathcal{D}_g$ is the union of S_g , as defined above, and of another set $\tilde{S}_g \subseteq \mathcal{D}_g$, consisting of those disks $D_k \in \mathcal{D}_g$ for which there exists $j \leq k$ such that (i) $D_j \in S_f$, and (ii) the entire run D_{j+1}, \dots, D_k is contained in \mathcal{D}_g . (so the disk D_j is not required to belong to the run).

A DFA interpretation. We can interpret the setup just described as a construction of a deterministic finite automaton \mathcal{K} , as follows; for the convenience of the reader, we include the following short glossary of the main notations used in this construction.

f or f_i (or g) - a face of $\mathcal{A}(\mathcal{D})$.
 F - string of n faces.
 F_k - a substring of F (of length τ).

φ or φ_i - a binary flag.
 Φ - string of n flags.
 Φ_k - a substring of Φ (of length τ).

σ - a pair (f, φ) of a face f and a flag φ .
 Σ - string of n pairs.
 Σ_k - a substring of Σ (containing τ pairs).

The states of \mathcal{K} are the aggregate states (f, S_f) , where f is a face of the corresponding disk arrangement \mathcal{A} and $S_f \subseteq \mathcal{D}_f$. The i -th ‘character’ in the string that \mathcal{K} has to process is the pair (g_i, φ_i) , where g_i is the face of \mathcal{A} that contains b_i , and φ_i is a flag indicating whether (a_1, b_i) is a reachable position of the two frogs (in the sense defined above, with respect to the whole trip). The transition from a state (f, S_f) on reading the pair (g_i, φ_i) is to (g_i, S_{g_i}) , where S_{g_i} is defined as above. The string that \mathcal{K} has to process to handle the current block A is thus the string of pairs $\Sigma = ((f_2, \varphi_2), \dots, (f_n, \varphi_n))$, where f_2, \dots, f_n are the (not

necessarily neighboring) faces of \mathcal{A} containing the corresponding actual points b_2, \dots, b_n of the B -sequence, and $\varphi_2, \dots, \varphi_n$ are the respective flags associated with b_2, \dots, b_n , as defined above.

The starting state of \mathcal{K} is the state (f_1, S_{f_1}) , where f_1 is the face containing b_1 , and where $S_{f_1} = \emptyset$ if $\varphi(b_1) = 0$, or S_{f_1} is the largest prefix of \mathcal{D} contained in \mathcal{D}_{f_1} if $\varphi(b_1) = 1$. Note that in the construction of \mathcal{K} we are not given that prefix — \mathcal{K} is defined in terms of A only. Furthermore, when \mathcal{K} does read the B -string Σ and reaches a state (f_i, S_{f_i}) it outputs a new flag $\varphi(b_i)$, which is 1 if $D_m \in S_{f_i}$ and is 0 otherwise. The points b_i with $\varphi(b_i) = 1$ are exactly those for which (a_m, b_i) is reachable in G_1 (from the beginning of the whole trip). In this context, we can think of \mathcal{K} as a *Moore machine* [21] — a finite-state transducer that associates an output value with each state. We can thus associate the output flag $\varphi(b_i)$ with the state (f_i, S_{f_i}) . The output flag $\varphi(b_i)$ will be used later, as an input for the next block (see Section 2.3).

As noted earlier, if A is the first block of the whole sequence, each flag of the input sequence Σ , except for the first one, is equal to zero. For the first position (a_1, b_1) of the first block, we assume that $b_1 \in D_1$; otherwise, as already mentioned, we abort the decision procedure right away, reporting that the Fréchet distance $\delta_{dF}(A, B)$ is greater than 1. We thus set, after verifying this constraint, $\varphi_1 = \varphi(b_1) = 1$.

Remark. The automaton \mathcal{K} is constructed from the block A only, without knowing anything about the sequence B . Consequently, for each face f of the arrangement \mathcal{A} , we need to prepare states (f, S_f) for each subset $S_f \subseteq \mathcal{D}_f$ that might arise via some sequence of stepping stones of the B -frog. As shown in Section 4, there are situations where the number of such feasible subsets may be exponential in $|\mathcal{D}_f|$ (that is, in m). This is why we need to take $m = c_2 \log n$, with c_2 sufficiently small, to control the size of \mathcal{K} and the time needed to construct it (so that they are both sublinear in n).

Constructing an efficient DFA. To obtain an overall procedure with subquadratic running time, we modify the construction of \mathcal{K} to obtain a somewhat more efficient automaton \mathcal{K}^* to handle a block A . There are two major improvements in the construction of \mathcal{K}^* . The first, whose detailed description is deferred to Section 2.2, is to construct \mathcal{K}^* in terms of the finer arrangement \mathcal{A}^* of the disks centered at all the $\Theta(\log^2 n)$ points of the A -sequence within the layer containing the current block. Informally, the reason for doing it (explained in detail in Section 2.2) is that it saves us the need to locate the B -points in each of the coarser block arrangements, a process that would be too expensive for our purpose. Nevertheless, so as not to throw at the reader the two improvements at the same time, we present here the construction of \mathcal{K}^* solely in terms of the current block arrangement \mathcal{A} and then modify it in the next subsection.

The second improvement aims to allow \mathcal{K}^* to process the B -dependent string Σ in a faster manner. Specifically, we modify \mathcal{K} so that each input character that it reads is a string of $c_3 \log n / \log \log n$ consecutive input characters of Σ , where $c_3 > 0$ is a sufficiently small constant, whose value will be determined later. That is, we partition the input string Σ of \mathcal{K} into $\mu = \Theta(\frac{n \log \log n}{\log n})$ substrings $\Sigma_1, \Sigma_2, \dots, \Sigma_\mu$ of size $\tau = c_3 \log n / \log \log n$ each; the last substring may be shorter. The states of \mathcal{K}^* are the same aggregate states (f, S_f) of \mathcal{K} . When \mathcal{K}^* is at state (f, S_f) and is given a substring $\Sigma_k = ((f_1, \varphi_1), \dots, (f_\tau, \varphi_\tau))$, it

moves to state (f_τ, S_{f_τ}) , where (f_τ, S_{f_τ}) is the state that \mathcal{K} would have reached from (f, S_f) after processing the input substring Σ_k character by character. (The subscripts used in the enumeration of the pairs of Σ_k start at 1 for the sake of simplicity. This involves a slight abuse of notation, because (f_1, φ_1) denotes here the first pair of Σ_k and not the first pair of the entire string Σ .)

Furthermore, a transition of \mathcal{K}^* from a state (f, S_f) to a state (f_τ, S_{f_τ}) as above, produces an output string $\Phi_k = (\varphi_1, \dots, \varphi_\tau)$, where φ_j is the output of \mathcal{K} when it reaches the state (f_j, S_{f_j}) (again, under the new enumeration convention). Recall that we regarded \mathcal{K} as a Moore machine, where the output flags φ_j are associated with the corresponding states (f_j, S_{f_j}) . However, here the state (f_τ, S_{f_τ}) that \mathcal{K}^* moves to after reading $((f_1, \varphi_1), \dots, (f_\tau, \varphi_\tau))$ cannot determine by itself the output string Φ_k , which requires knowledge of the full sequence $((f_1, \varphi_1), \dots, (f_\tau, \varphi_\tau))$ that led \mathcal{K}^* to (f_τ, S_{f_τ}) . More specifically, the flags comprising Φ_k are determined by the states (f_j, S_{f_j}) that \mathcal{K} traversed on the way to (f_τ, S_{f_τ}) . To avoid having to look at each intermediate state (f_j, S_{f_j}) separately, we observe that all these states are implicitly encoded in the transition *edge* of \mathcal{K}^* that takes us from (f, S_f) to (f_τ, S_{f_τ}) upon reading Σ_k . We can therefore regard \mathcal{K}^* as a *Mealy machine* [20] — a finite-state transducer that associates an output value with each transition edge.

The rest of the description of \mathcal{K}^* remains the same as that of \mathcal{K} .

In the following, we describe how to construct \mathcal{K}^* so that a state transition can be carried out in constant time. (A full description of the construction of \mathcal{K}^* will be given in the next subsection.) As is shown later, executing a transition of \mathcal{K}^* in constant time is essential for obtaining the subquadratic running time of the whole optimization procedure.

To construct \mathcal{K}^* , we build the transition table T , according to the rules stated above. Since T is constructed independently of the input string Σ , we must prepare, for each state (f, S_f) of \mathcal{K}^* , all possible transitions to a new state. That is, given a state (f, S_f) we store, for each possible input substring Σ_k of length τ , the state (g, S_g) that \mathcal{K} moves to after processing Σ_k (assuming that \mathcal{K} was in state (f, S_f) just before reading Σ_k). To be more precise, we prepare the transition table T of \mathcal{K}^* as a collection of arrays $L_{(f, S_f)}$, one array for each state (f, S_f) of \mathcal{K}^* . The array $L_{(f, S_f)}$ of a fixed state (f, S_f) is defined so that, for each index j encoding a substring Σ_k (details of the encoding are provided next), $L_{(f, S_f)}[j]$ is the pair $((g, S_g), \Phi_k)$, where (g, S_g) is the state that \mathcal{K}^* moves to after processing Σ_k (assuming that \mathcal{K} was in state (f, S_f) just before reading Σ_k), and Φ_k is the output substring of flags that corresponds to this transition.

To complete the description of T , we now describe a simple encoding scheme that converts each string $\Sigma_k = ((f_1, \varphi_1), \dots, (f_\tau, \varphi_\tau))$ of τ pairs into an integer $e(\Sigma_k)$ of $O(\log n)$ bits. To do so, each face f of the arrangement of the disks of A is given an integer label $e(f)$ in the range $(0, \dots, c \log^4 n)$, for an appropriate absolute constant c . (We will later explain, as part of the full description of the construction of \mathcal{K}^* , why we use this range and how to generate the labels efficiently.) Clearly, at most $\beta = \log(c \log^4 n) = c' \log \log n$ bits are needed for such a label, for another absolute constant c' (close to 4). We now put

$$e(\Sigma_k) = \sum_{i=1}^{\tau} e(f_i) \cdot 2^{\beta(i-1)+\tau} + \sum_{i=1}^{\tau} \varphi_i \cdot 2^{i-1}, \quad (1)$$

and note that $e(\Sigma_k)$ does indeed consist of only $\tau(\beta + 1) = O(\log n)$ bits. Clearly, this is a one-to-one encoding.

With this setup, each state transition can be executed in constant time. Specifically, when \mathcal{K}^* is in state (f, S_f) and is given the encoding $e(\Sigma_k)$ of an input substring Σ_k , we follow a pointer to the array $L_{(f, S_f)}$ and retrieve its entry $L_{(f, S_f)}[e(\Sigma_k)]$ in constant time. This gives us the next state (g, S_g) and the corresponding output bitstring Φ_k . Hence, the execution of \mathcal{K}^* , when given $O\left(\frac{n \log \log n}{\log n}\right)$ substrings as above, takes $O\left(\frac{n \log \log n}{\log n}\right)$ time. This cost excludes the computation of the indices $e(\Sigma_k)$, which will be discussed in the next subsection.

The size of (number of entries in) T is the number of states of \mathcal{K}^* , multiplied by the number of possible input substrings for \mathcal{K}^* . The latter number is $2^{(\beta+1)\tau} \leq 2^{c'' \log n}$, where c'' is proportional to c_3 , which we choose sufficiently small so as to have $c'' < 1/4$, say. The number of states of \mathcal{K}^* is $O(m^2 2^m)$, where $m = c_2 \log n$ is the size of a block.¹ There are $O(m^2)$ faces in the disk arrangement, and, in view of the construction given in Section 4, we use the pessimistic bound of 2^m on the number of possible subsets S_f for any fixed face f . Choosing c_2 sufficiently small, we can ensure that the number of states of \mathcal{K}^* is at most $O(n^{1/4})$, say. Hence the size of T is $O(n^{1/2}) = O(n \log \log n / \log n)$, and it can be built within the same asymptotic time bound.

2.2 Handling a layer of A

In order to make the whole procedure efficient, we need to construct quickly the encodings of the input strings for the automata of the blocks of A . Note that we cannot even afford linear (i.e., $O(n)$) time for this preparation for each block, because this would result in the overall bound $O(mn / \log n)$ for the running time of the decision procedure, which, multiplied by the number $O(\log n)$ of binary search steps, would yield $O(mn)$ overall running time, which defeats our goal of obtaining a subquadratic solution.

This is the reason for using a two-stage partitioning of A , first into layers of size $c_1 \log^2 n$ each, and then into blocks of size $c_2 \log n$ each — The preparation of the strings is done mainly at the layer level, thereby making the cost sublinear for each block.

Here are the details of this preprocessing step. Fix a layer \bar{A} of A , which contains $t = \Theta(\log n)$ blocks, of size $c_2 \log n$ each, which we enumerate as A_1, \dots, A_t . As before, the last point of A_i , for $1 \leq i < t$, is the first point of A_{i+1} . We process A_1, \dots, A_t in order, in much the same way as described in Section 2.1, except that some of the preparatory steps are grouped together, and take place during the preprocessing of \bar{A} .

In more detail, we first construct the arrangement $\bar{\mathcal{A}} = \bar{\mathcal{A}}(\bar{\mathcal{D}})$, where $\bar{\mathcal{D}}$ is the set of $c_1 \log^2 n$ unit disks centered at the points of \bar{A} ; the number of faces of $\bar{\mathcal{A}}$ is at most $c \log^4 n$, for an appropriate constant c (the same constant appearing in the encoding in the previous subsection). We preprocess $\bar{\mathcal{A}}$ for efficient point location, using any of the standard techniques, in $O(\log^4 n \log \log n)$ time. Fix a block A_j of \bar{A} , and note that each face f of $\bar{\mathcal{A}}$ is a subface of a face $f^{(j)}$ of the arrangement \mathcal{A}_j of the disks centered at the points of A_j . We find these correspondences by preprocessing each \mathcal{A}_j for fast point location, and then, for each face f of $\bar{\mathcal{A}}$ we pick an arbitrary point in f and locate it in \mathcal{A}_j , thereby obtaining $f^{(j)}$. In this way each face of $\bar{\mathcal{A}}$ stores t pointers to its “super-faces” $f^{(j)}$, for $j = 1, \dots, t$.

¹The first improvement in \mathcal{K}^* , deferred to Section 2.2, will cause the number of states to increase to $O(m^4 2^m)$, which will have negligible effect on the performance of the algorithm; see below for details.

Next, for each point b_i of the B -sequence, we locate the face f_i of $\bar{\mathcal{A}}$ containing b_i , using the point location structure. This takes $O(n \log \log n)$ time. We obtain a sequence $F = (f_1, f_2, \dots, f_n)$ of faces of $\bar{\mathcal{A}}$, and we partition it into μ subsequences F_1, \dots, F_μ , each consisting of τ consecutive faces, where $\mu = \Theta(n \log \log n / \log n)$ and $\tau = c_3 \log n / \log \log n$, as in the preceding subsection.

Now comes the other improvement in the construction of the block-automata \mathcal{K}^* considered in Section 2.1. Specifically, since the number of faces of $\bar{\mathcal{A}}$ is at most $c \log^4 n$, we label each face f of $\bar{\mathcal{A}}$ by an integer $e(f)$ in the range $(0, \dots, c \log^4 n)$. For each of the μ subsequences F_k of F , say $F_k = (f_1, \dots, f_\tau)$, we compute the “partial” index (cf. (1))

$$e_0(F_k) = \sum_{i=1}^{\tau} e(f_i) \cdot 2^{\beta(i-1)+\tau}. \quad (2)$$

Note that, given the labels $e(f_i)$, $e_0(F_k)$ can be computed by $O(\tau)$ additions and multiplications (or, rather, shifts). In addition, note that this index is common to all the blocks of $\bar{\mathcal{A}}$; we stress again that each such partial index is computed only once within the layer $\bar{\mathcal{A}}$.

Now fix a block A_j of $\bar{\mathcal{A}}$, and consider the construction of its automaton \mathcal{K}_j^* . Except for the fact that the faces of $\bar{\mathcal{A}}$ that we use here are smaller than the respective faces of the block arrangement \mathcal{A}_j , the states (f, S_f) and the transition rules for \mathcal{K}_j^* are very similar to those used in Subsection 2.1. More specifically, each face f_0 of \mathcal{A}_j is now the union of some faces of $\bar{\mathcal{A}}$. Every state of the form (f_0, S_{f_0}) that we had before is now copied, for each face $f \subseteq f_0$ of $\bar{\mathcal{A}}$, to a state (f, S_{f_0}) . A similar copying is applied to the transition rules. That is, consider first the non-compacted automaton \mathcal{K}_j . If it is at a state (f, S_f) and reads a pair (g, φ) , where f and g are now faces of $\bar{\mathcal{A}}$, we apply the same transition rule that the original \mathcal{K}_j obeys when it is at state (f_0, S_{f_0}) and reads (g_0, φ) , where f_0 (resp., g_0) is the face of \mathcal{A}_j containing f (resp., g). We now obtain the new version of \mathcal{K}_j^* from the new version of \mathcal{K}_j in the same manner as above. That is, when \mathcal{K}_j^* is at state (f, S_f) and reads a substring $\Sigma_k = ((f_1, \varphi_1), \dots, (f_\tau, \varphi_\tau))$ of Σ , where now f, f_1, \dots, f_τ are faces of $\bar{\mathcal{A}}$, it moves to the state (f_τ, S_{f_τ}) obtained by running the new \mathcal{K}_j on the pairs of Σ_k one by one.

The total time for computing the μ indices $e_0(F_k)$ is linear in n . This is tolerable since we carry out this computation only once for the entire layer $\bar{\mathcal{A}}$. However, each of the subsequences Σ_k that we feed into the various block automata \mathcal{K}_j^* has a second “component” that depends on the input flags at the first row of the respective block A_j . Specifically, each Σ_k is of the form $((f_1, \varphi_1), \dots, (f_\tau, \varphi_\tau))$, which we can represent by the pair (F_k, Φ_k) , where $F_k = (f_1, \dots, f_\tau)$ and $\Phi_k = (\varphi_1, \dots, \varphi_\tau)$. The subsequences F_k are computed once, at the layer level, and do not change from block to block, but the subsequences Φ_k do depend on the blocks. In terms of the encoding in (1) we have

$$e(\Sigma_k) = e_0(F_k) + e_0(\Phi_k), \quad (3)$$

where

$$e_0(\Phi_k) = \sum_{i=1}^{\tau} \varphi_i \cdot 2^{i-1} \quad (4)$$

is simply the bitstring consisting of the flags in Φ_k .

We can easily construct the automata \mathcal{K}_j^* in such a way that the output of each transition is the encoding $e_0(\Phi_k)$ of the corresponding sequence Φ_k . Assuming that this is the case, we process a block A_j as follows. Let Φ_1, \dots, Φ_μ denote the output flag subsequences from the execution of the preceding automaton \mathcal{K}_{j-1}^* (or from the execution of the last automaton in the preceding layer, or from the initialization of the entire procedure). By assumption, we are actually given the encodings $e_0(\Phi_1), \dots, e_0(\Phi_\mu)$ (the computation of these bitstrings during initialization is trivial and inexpensive), and we substitute them in (3) to obtain $e(\Sigma_1), \dots, e(\Sigma_\mu)$. This computation takes $O(\mu) = O(n \log \log n / \log n)$ time for each block, for a total of $O(n \log \log n)$ time for the whole layer. We now run (the modified automaton) \mathcal{K}_j^* on the string $(e(\Sigma_1), \dots, e(\Sigma_\mu))$ and obtain the output sequence $e_0(\Phi'_1), \dots, e_0(\Phi'_\mu)$, where $\Phi'_1, \dots, \Phi'_\mu$ are the flag subsequences output by the state transitions of \mathcal{K}_j^* , which are the input for the next automaton.

The analysis in Section 2.1 shows that, with an appropriate choice of the constants c_1, c_2, c_3 , the construction of the automata \mathcal{K}_j^* , for $j = 1, \dots, t$, takes a total of $O(n)$ time (in fact, much smaller if we so wish). Processing a single block costs $O(n \log \log n / \log n)$ time (see Section 2.1 and the preceding paragraph). Since \bar{A} contains $\Theta(\log n)$ blocks,² the total cost for processing \bar{A} is $O(n \log \log n)$. (This includes the cost of the point location stage within \bar{A} , which is also $O(n \log \log n)$.) In conclusion, processing a single layer, including the processing of each of its blocks, takes a total of $O(n \log \log n)$ time.

The space required for this procedure is linear in n , since we need to store the subsequences of faces of \bar{A} , which are used as input for each \mathcal{K}_i^* . The space used for handling a block A_i of \bar{A} is sub-linear in n (see Section 2.1), and can be freed after processing A_i . Hence, the total space required for processing \bar{A} is still linear in n .

2.3 The overall procedure

To obtain an overall algorithm with subquadratic time, we partition the original sequence A into $\Theta(m / \log^2 n)$ layers $\bar{A}_1, \bar{A}_2, \dots$, each (except possibly for the last one) consisting of $c_1 \log^2 n$ points, and so that the last point of \bar{A}_i is the first point of \bar{A}_{i+1} for each i . We then process $\bar{A}_1, \bar{A}_2, \dots$ in succession.

To process a layer \bar{A}_i , we use the procedure of Section 2.2. If \bar{A}_i is not the last layer of A , we use the output sequence Φ_1, \dots, Φ_μ of \bar{A}_i as input for \bar{A}_{i+1} (as described in Section 2.2). Otherwise, \bar{A}_i is the last layer of A , and we use the last flag φ_τ of the last subsequence Φ_μ to determine the outcome of the decision process — if $\varphi_\tau = 1$ we report that $\delta_{dF}(A, B) \leq \delta$; otherwise $\delta_{dF}(A, B) > \delta$.

Processing each layer \bar{A}_i of A takes $O(n \log \log n)$ time, so processing the $\Theta(m / \log^2 n)$ layers, takes $O(mn \log \log n / \log^2 n)$ time. The space required for handling a layer \bar{A}_i of A is linear in n (see Section 2.2), and it can be freed after handling \bar{A}_i . Hence, the space required by the decision procedure is only $O(n + m)$ (we need $O(m)$ space for storing A).

Hence, we obtain the following intermediate result.

Theorem 2.1. *Given two sequences A, B of stepping stones, of respective sizes m and n , with $m \leq n$, and a parameter $\delta > 0$, we can decide, using $O\left(\frac{mn \log \log n}{\log^2 n}\right)$ time and $O(n + m)$*

²This step in the analysis is the reason for restricting the size of a layer to $\Theta(\log^2 n)$ points of A , that is, to $\Theta(\log n)$ blocks.

space, whether $\delta_{dF}(A, B) \leq \delta$.

Remark. The above procedure determines whether $\delta_{dF}(A, B) > \delta$ or $\delta_{dF}(A, B) \leq \delta$. In the latter situation, there is no need to discriminate between $\delta_{dF}(A, B) < \delta$ and $\delta_{dF}(A, B) = \delta$, since this could easily be done upon termination of the binary search, as described in Section 3, by comparing two consecutive critical values of δ reached at the end of the search. See Section 3 for more details.

3 Solving the optimization problem

We use the decision procedure in Section 2 to solve the optimization problem, as follows. First note that the critical values of δ , in which an edge is added to the graph G_δ (as δ increases), are the pairwise distances between a point of A and a point of B . Hence, it suffices to perform a binary search over all possible mn such distances, and execute the decision procedure in each step of the search. At each such step, the corresponding pairwise distance is the l -th smallest pairwise distance in $A \times B$ for some value of l . We can find this distance, e.g., using a variant of one of the algorithms of Agarwal et al. [1], which runs in time close to $O(n^{3/2})$. This algorithm can easily be adapted to the “bichromatic” scenario, where we consider only distances between the pairs in $A \times B$ (as opposed to finding distances between the points of a single set). More specifically, we use a variant of the simpler (sequential) decision procedure of [1]. We partition the set A into $\lceil m/n^{1/2} \rceil$ smaller subsets, each of size at most $n^{1/2}$, and operate on each subset independently, coupled with the whole B . In processing such a subset A_i , we construct the arrangement of the disks of radius δ centered at the points of A_i , and locate the points of B in this arrangement, exactly as in [1]. Altogether, this yields the number of pairs in $A \times B$ at distance at most δ , which is what the decision procedure needs. The overall cost of this procedure is $O(n^{3/2} \log n)$. Finally, we solve the optimization version of the distance selection algorithm using parametric searching, increasing the running time to $O(n^{3/2} \log^3 n)$. This running time is subsumed by the cost of the decision procedure of Section 2.³

Since we call the decision procedure $O(\log n)$ times during the search, we obtain the following main result of the paper.

Theorem 3.1. *Given two sequences A, B of stepping stones, of respective sizes m and n , with $m \leq n$, we can compute the discrete Fréchet distance between A and B in $O\left(\frac{mn \log \log n}{\log n}\right)$ time and $O(n + m)$ space.*

4 An exponential lower bound on the number of states

An interesting question that pops up right away in the design of the algorithm is how large can \mathcal{K}^* be. That is, how many aggregate states (and transition rules) can one have.

³Although there are more efficient algorithms for distance selection, which run in close to $O(n^{4/3})$ time [1, 18], this simple-minded solution suffices for our purpose, and it has the advantage that it only uses linear storage.

Unfortunately, the following construction shows that this number can be exponential in m in the worst case.

The construction, depicted in Figure 3, uses an even number of disks; with a slight abuse of notation, we denote their number by $2m$. Enumerate the disks as D_1, D_2, \dots, D_{2m} and their respective centers as a_1, a_2, \dots, a_{2m} . All these centers lie on the x -axis in the right-to-left order $a_1, a_3, \dots, a_{2m-1}, a_2, a_4, \dots, a_{2m}$. The centers of the even-indexed disks (red disks for short) are sufficiently close to each other, so that these disks have a large common intersection. The odd-indexed disks (blue disks for short) are placed so that, for each $k = 1, \dots, m$, D_{2k-1} intersects D_{2k} (in a small cap) but is disjoint from D_{2k+2} (the second condition is vacuous for $k = m$).

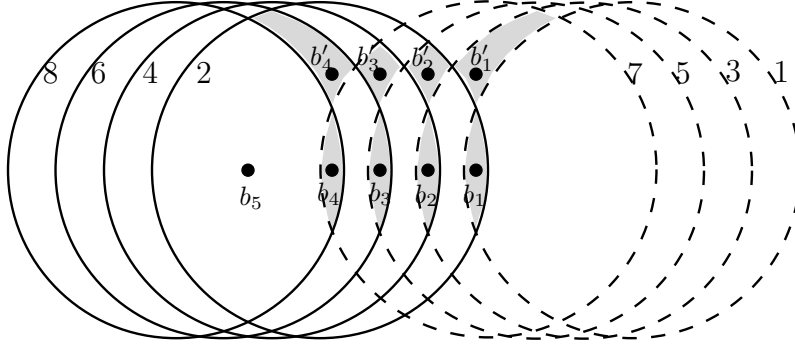


Figure 3: A configuration of disks with an exponential number of states. The red disks are drawn solid and the blue disks are drawn dashed.

We next place $2m + 1$ points $b_1, b'_1, b_2, b'_2, \dots, b_m, b'_m, b_{m+1}$ (or, rather, select $2m + 1$ corresponding faces $f_1, f'_1, f_2, f'_2, \dots, f_m, f'_m, f_{m+1}$ of the resulting arrangement of the $2m$ disks). For each $i = 1, \dots, m$, we take f_i to be the cap $D_{2i-1} \cap D_{2i}$ (by construction, and as shown in the figure, these are indeed faces of the arrangement). We take f'_i to be the face lying directly above f_i , so that in order to go from f_i to f'_i we need to exit the two disks D_{2i-1} and D_{2i} (and not to cross the boundary of any other disk). Finally, we take f_{m+1} to be the intersection face of all the red (even-indexed) disks.

We regard (a_1, b_1) as the starting position of the frogs, where b_1 is any point in f_1 and a_1 is the center of D_1 , and the goal position is (a_{2m}, b_{m+1}) , where b_{m+1} is any point in f_{m+1} and a_{2m} is the center of D_{2m} .

By construction, $\mathcal{D}_{f_{m+1}}$ consists of all the m red disks. We claim that for every subset $S \subseteq \mathcal{D}_{f_{m+1}}$, (f_{m+1}, S) is a valid state, obtaining the asserted exponential number of states. To be more precise, the claim is that for any such S we can construct a sequence $B = B_S$ of points, which (i) starts at b_1 and ends at b_{m+1} , (ii) contains all the points b_1, b_2, \dots, b_{m+1} (in this order), and (iii) contains some of the points b'_1, \dots, b'_m , so that if it contains b'_j then b'_j appears between b_j and b_{j+1} . The sequence B_S has the property that for any $D \in S$, as the B -frog moves through the sequence B_S , the A -frog can execute a sequence of corresponding moves, so that it reaches at the end the center of D , and this cannot be achieved (for the same sequence B_S) for any $D \notin S$. For simplicity, we only specify the sequence of faces of \mathcal{A} containing the points of B , rather than the points themselves (although the figure depicts the points too).

So let $S \subseteq \mathcal{D}_{f_{m+1}}$ be given. We associate with S the following sequence F_S of faces. We

start with the subsequence $(f_1, f_2, \dots, f_m, f_{m+1})$ and, for each D_{2k} *not in* S , we insert f'_k into F_S , between f_k and f_{k+1} . Figuratively, the corresponding sequence B_S , which proceeds from right to left, is a mixture of sharp vertical detours (corresponding to red disks not in S) and of short horizontal moves (for red disks in S).

We next argue that F_S does indeed generate the state (f_{m+1}, S) . Consider a red disk D_{2k} not in S . When the B -frog follows the detour from f_k to f'_k and then to f_{k+1} , it leaves D_{2k-1} and D_{2k} and then re-enters D_{2k} (and D_{2k+2}). The maximal run of disks which ends at D_{2k} and is contained in $\mathcal{D}_{f_{k+1}}$, includes D_{2k} only, since $D_{2k-1} \notin \mathcal{D}_{f_{k+1}}$. In addition, D_{2k} does not belong to $\mathcal{D}_{f'_k}$, so in particular $D_{2k} \notin S_{f'_k}$. Hence, $D_{2k} \notin S_{f_{k+1}}$, because there is no valid transition (in this setup) from $(f'_k, S_{f'_k})$ to $(f_{k+1}, S_{f_{k+1}})$ such that $D_{2k} \in S_{f_{k+1}}$ (see the rules for a valid transition in Section 2.1). From this point on, the path is fully outside of D_{2k-1} , so, as easily verified by induction, D_{2k} will not appear in any of the following states, including the state $(f_{m+1}, S_{f_{m+1}})$, as claimed. (The reader might wish to interpret this argument in terms of the actual moves of the frogs.)

Consider next a red disk D_{2k} that belongs to S . It suffices to show that when the B -frog reaches f_k , the A -frog could have executed a sequence of preceding moves that gets it to the center of D_{2k} ; this is because, from this point on, the B -frog remains inside D_{2k} (note that, by construction, we do not execute the detour via f'_k), so the A -frog simply has to stay put at the center of D_{2k} and wait for the end of the sequence of moves of the B -frog.

Note that f_1 is contained in all blue disks and in D_2 . In particular, this implies the asserted property for $k = 1$: The A -frog goes from the center of D_1 to the center of D_2 before the B -frog moves, and stays there till the end. In general, f_j is contained in the blue disks $D_{2j-1}, D_{2j+1}, \dots, D_{2m-1}$ and in the red disks D_2, D_4, \dots, D_{2j} . What the A -frog needs to do is to ensure that, for each $j < k$, it lies at the center of D_{2j+1} by the time the B -frog gets to f_j . This is easily argued by induction on j . The A -frog can do this for $j = 1$, because f_1 lies in D_1, D_2, D_3 . For larger values of j , assume that the A -frog is at the center of D_{2j-1} when the B -frog is at f_{j-1} . If the path goes straight to f_j , it exits D_{2j-3} and then enters D_{2j} . Since the A -frog is at the center of D_{2j-1} , it can now move to the center of D_{2j} and then to the center of D_{2j+1} , as desired. If the path goes to f_j via f'_{j-1} , it exits D_{2j-3} and D_{2j-2} , then re-enters D_{2j-2} and then enters D_{2j} . However, since the A -frog is already at the center of D_{2j-1} , these additional exit and re-entry are irrelevant for it, and it can now move to the center of D_{2j+1} as above. Finally, when the B -frog moves to f_k , the A -frog, which is now at the center of D_{2k-1} , moves to the center of D_{2k} and stays there. This completes the argument.

Remark. It is a challenging open problem to circumvent this exponential lower bound on the number of possible states. Of course, we have exponentially many states because of the existence of exponentially many possible B -sequences. Is it possible, for example, to reduce the number of states significantly by some sort of examination of the specific input B -sequence? As already remarked, the existence of potentially exponentially many states is the major bottleneck for the efficiency of the algorithm. In the same vein, it would be interesting to find properties of the sequences A , B that guarantee that the number of aggregate states is much smaller. In a sense, this would hopefully subsume (so far, for the discrete and semi-continuous cases only) the earlier studies involving special classes of curves and/or sequences [4, 5, 13].

5 Discussion and open problems

We obtained an algorithm for computing the discrete Fréchet distance between two sets of points, which runs in subquadratic time. A natural open problem that arises right away is whether this algorithm can be extended to compute the continuous Fréchet distance between two polygonal curves in subquadratic time. Even solving the semi-continuous Fréchet distance problem in subquadratic time might be interesting at this point. It is also interesting to know if this time bound, which is still rather close to quadratic, can be further reduced (see the remark at the end of the preceding section).

References

- [1] P. K. Agarwal, B. Aronov, M. Sharir and S. Suri, Selecting distances in the plane, *Algorithmica* 9(5) (1993), 495–514.
- [2] H. Alt, The computational geometry of comparing shapes, *Efficient Algorithms: Essays Dedicated to Kurt Mehlhorn on the Occasion of His 60th Birthday*, Springer-Verlag (2009), 235–248.
- [3] H. Alt and M. Godau, Computing the Fréchet distance between two polygonal curves, *Internat. J. Comput. Geom. Appl.* 5 (1995), 75–91.
- [4] H. Alt, C. Knauer and C. Wenk, Comparison of distance measures for planar curves, *Algorithmica* 38(1) (2004), 45–58.
- [5] B. Aronov, S. Har-Peled, C. Knauer, Y. Wang and C. Wenk, Fréchet distance for curves, revisited, *Proc. 14th Annu. European Sympos. Algorithms* (2006), 52–63.
- [6] I. Baran, E. D. Demaine and M. Patrascu, Subquadratic algorithms for 3SUM, *Algorithmica* 50(4) (2008), 584–596.
- [7] S. Brakatsoulas, D. Pfoser, R. Salas and C. Wenk, On map-matching vehicle tracking data, *Proc. 31st Intl. Conf. Very Large Data Bases* (2005), 853–864.
- [8] K. Buchin, M. Buchin, C. Knauer, G. Rote and C. Wenk, How difficult is it to walk the dog? *Proc. 23rd Euro. Workshop Comput. Geom.* (2007), 170–173.
- [9] D. Chen, A. Driemel, L. J. Guibas, A. Nguyen and C. Wenk, Approximate map matching with respect to the Fréchet distance, *Proc. 7th Workshop on Algorithm Engineering and Experiments* (2011), 75–83.
- [10] A. F. Cook, A. Driemel, S. Har-Peled, J. Sherette and C. Wenk, Computing the Fréchet distance between folded polygons, *Proc. Lect. Notes in Comp. Sci.* (2011), 267–278.
- [11] M. Crochemore, G. M. Landau and M. Ziv-Ukelson, A subquadratic sequence alignment algorithm for unrestricted scoring matrices, *SIAM J. Comput.* 32 (2003), 1654–1673.
- [12] A. Driemel and S. Har-Peled, Jaywalking your dog: computing the Fréchet distance with shortcuts, *Proc. 23rd Annu. ACM-SIAM Sympos. Discrete Algorithms* (2012), 318–337.

- [13] A. Driemel, S. Har-Peled and C. Wenk, Approximating the Fréchet distance for realistic curves in near linear time, *Proc. 26th Annu. ACM Sympos. Comput. Geom.* (2010), 365–374.
- [14] T. Eiter and H. Mannila, Computing discrete Fréchet distance, Technical Report CD-TR 94/64, Christian Doppler Laboratory for Expert Systems, TU Vienna, Austria, 1994.
- [15] A. Gajentaan and M. H. Overmars, On a class of $O(n^2)$ problems in computational geometry, *Comput. Geom. Theory Appl.* 5(3) (1995), 165–185.
- [16] S. Har-Peled and B. Raichel, The Fréchet distance revisited and extended, *Proc. 27th Annu. ACM Sympos. Comput. Geom.* (2011), 448–457.
- [17] D. Hermelin, G. M. Landau, S. Landau and O. Weimann, A unified algorithm for accelerating edit-distance computation via text-compression, *Proc. 26th Sympos. Theoret. Asp. Comp. Sci.* (2009), 529–540.
- [18] M. J. Katz and M. Sharir, An expander-based approach to geometric optimization, *SIAM J. Comput.* 26(5) (1997), 1384–1408.
- [19] W. J. Masek and M. S. Paterson, A faster algorithm computing string edit distances, *J. Comput. Sys. Sci.* 20(1) (1980), 18–31.
- [20] G. H. Mealy, A method to synthesizing sequential circuits, *Bell Systems Technical Journal.* 34 (1955), 1045–1079.
- [21] E. F. Moore, Gedanken-experiments on sequential machines, *Automata Studies, Annals of Mathematical Studies*, Princeton University Press, Princeton, NJ, 34 (1956), 129–153.