Computing the Discrete Fréchet Distance in Subquadratic Time*

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

Abstract

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

The discrete Fréchet distance replaces the dog and its owner by a pair of frogs that can only reside on $n$ and $m$ specific stones on the curves $A$ and $B$, respectively. These frogs hop from one stone to the next without backtracking, and the discrete Fréchet distance is the minimum length of a “leash” that connects the frogs and allows them to execute such a sequence of hops. It can be computed in quadratic time by a straightforward dynamic programming algorithm.

We present the first subquadratic algorithm for computing the discrete Fréchet distance between two sequences of points in the plane. Assuming $m \leq n$, the algorithm runs in $O\left(\frac{m \log n}{\log \log n}\right)$ time, in the standard RAM model, using $O(n)$ 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 automaton.

1 Introduction

Let $A = (a_0, \ldots, a_{m-1})$ and $B = (b_0, \ldots, b_{n-1})$ be two sequences of $m$ and $n$ points, respectively, in the plane. The discrete Fréchet distance $dF(A, B)$ between $A$ and $B$ is defined as follows. Fix a distance $\delta > 0$ and consider a directed graph $G_{\delta}(A, B) = (A \times B, E_A \cup E_B \cup E_{AB})$, where the edge sets $E_A$, $E_B$, and $E_{AB}$ are defined as follows:

$$E_A = \{(a_i, b_j) \mid \|a_i - b_j\|, \|a_{i+1} - b_j\| \leq \delta\},$$

$$E_B = \{(a_i, b_j) \mid \|a_i - b_j\|, \|a_i - b_{j+1}\| \leq \delta\},$$

$$E_{AB} = \{(a_i, b_j) \mid \|a_i - b_j\|, \|a_{i+1} - b_{j+1}\| \leq \delta\}.$$

Here we assume $\| \cdot \|$ to be the Euclidean norm. Then $dF(A, B)$ is the smallest $\delta > 0$ for which $(a_m, b_n)$ is reachable from $(a_1, b_1)$ in $G_{\delta}$. 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 by a rope of length $\delta$, and are initially placed at $a_1$ and $b_1$, respectively. At each move, either one of the frogs jumps from its current stone to the next one and the other stays at its current stone, or both of them jump simultaneously from their current stones to the next ones. Furthermore, such a jump is allowed only if the distance between the two frogs before the jump as well as after the jump is at most $\delta$. The edges in $E_A$ (resp., $E_B$) correspond to the case when the $A$-frog (resp., $B$-frog) jumps and the other stays at its current stone, and the edges in $E_{AB}$ correspond to the case when both frogs jump simultaneously. Then $dF(A, B)$ is the smallest $\delta > 0$ for which there exists a sequence of jumps that gets the frogs to $a_m$ and $b_n$, respectively.

The discrete Fréchet distance is a simpler version of the more standard (continuous) Fréchet distance. Informally, consider a person and a dog connected by a leash, each walking along a 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 \( \alpha : [0, 1] \to [0, 1] \), such that \( \alpha(0) = 0 \) and \( \alpha(1) = 1 \). The Fréchet distance \( F(f, g) \) between two curves \( f \) and \( g \) is then defined as follows:

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

where \( \| \cdot \| \) is the underlying norm (typically, as above, the Euclidean norm), and \( \alpha \) and \( \beta \) are reparameterizations of \([0, 1]\).

One can also consider the semi-continuous Fréchet distance, between a continuous curve \( g \) and a sequence \( A = (a_0, \ldots, a_{m-1}) \) of discrete points, where we want to find the smallest \( \delta \) for which \( g \) can be partitioned into \( m \) arcs \( (g_0, \ldots, g_{m-1}) \), so that, for each \( i \), the distance from \( a_i \) to every point in \( g_i \) is at most \( \delta \). Informally, \( \delta \) is the length of the shortest leash with which a person traversing \( g \) can walk a frog jumping through the points of \( A \) (again, with neither of them allowed to backtrack).

**Background.** Motivated by a variety of applications, the Fréchet distance has been studied extensively in computational geometry as a useful measure for comparing two curves; applications include dynamic time-warping [27], speech recognition [29], signature verification [34], matching of time series in databases [28], map-matching of vehicle tracking data [8, 14, 37], and analysis of moving objects [9, 10]. The simpler variant of the discrete Fréchet distance arises when we replace each of the input curves by a sequence of densely sampled points, and regard the resulting discrete distance as a good approximation of the actual continuous distance. In particular, the discrete Fréchet distance was considered for applications such as coastline matching [32], handwriting recognition [36], and protein structure alignment [25, 38]. The Fréchet distance and their variants are more useful for comparing curves than other measures such as the Hausdorff distance because they take into account the ordering of the points along the curves. See Figure 1 for an example of non-similar pair of curves (in the Fréchet sense) which have small Hausdorff distance.

![Figure 1: A pair of curves that are similar under the Hausdorff distance, but not similar under the Fréchet distance.](image)

Eiter and Mannila [20] showed that the discrete Fréchet distance in the plane can be computed, using dynamic programming, in \( O(mn) \) time. Aronov et al. [6] presented a near-linear-time \((1+\varepsilon)\)-approximation algorithm for computing the discrete Fréchet distance between the vertices of two backbone curves. These curves are required to have edges whose lengths are close to 1, and a positive constant lower bound on the minimal distance between any pair of vertices; they model, e.g., the backbone chains of proteins. Alt and Godau [4] showed 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. Buchin et al. [11] gave a lower bound of \( \Omega(n \log n) \) time for the problem of deciding whether the Fréchet distance between two curves is smaller than or equal to a given value. Their bound holds in the algebraic computation tree model allowing arithmetic operations and tests. They showed that this lower bound holds for the discrete Fréchet distance as well.

A subquadratic algorithm for computing the continuous or discrete Fréchet distance has remained elusive so far. In fact, Alt [3] recently conjectured that the decision subproblem of the (continuous) Fréchet distance problem was 3SUM-hard [22]. We note that no subquadratic algorithm (in \( m \) and \( n \), with any reasonable dependence on \( \varepsilon \)) is known even for computing a \((1+\varepsilon)\)-approximation of either variant of the Fréchet distance between two arbitrary curves (or point sequences), with no restrictions on their shapes. However, similar to the discrete case mentioned earlier, subquadratic algorithms are known for a few restricted classes of curves [5, 6, 19]. For example, Driemel et al. [19] provided a \((1+\varepsilon)\)-approximation algorithm for \( c\)-packed curves in \( \mathbb{R}^d \) that runs in \( O(cn/\varepsilon + cn \log n) \) time—a curve \( \pi \) is called \( c\)-packed if the total length of \( \pi \) inside any disk is bounded by \( c \) times the radius of the disk. See [8, 11, 18, 23, 30] for other variants of the Fréchet distance.

Recently, Maheshwari et al. [31] solved the decision version of the partial Fréchet distance in \( O(mn) \) time, improving upon a result of Alt and Godau [4]. In this problem, given two polygonal curves \( f \) and \( g \) and a parameter \( \delta > 0 \), one needs to decide whether there is a subcurve \( h \subseteq f \) whose Fréchet distance to \( g \) is at most \( \delta \). 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. This problem was considered by Wenk et al. [8, 37], by Buchin et al. [11] and, more recently by Har-Peled and Raichel [23]. Other variants include the Fréchet distance between folded polygons (which are non-flat surfaces) [15], and the Fréchet distance with outliers [18].

**Our results.** Given two sequences \( A \) and \( B \) of \( m \) and \( n \) points, respectively, in \( \mathbb{R}^2 \), we present an algorithm
for computing the discrete Fréchet distance between $A$ and $B$. Assuming $m \leq n$, the algorithm runs in $O(mn \log \log n)$ time in the standard RAM model and uses $O(n)$ storage.

We note that similar improvements have been obtained for quadratic solutions of several other problems. For instance, Baran et al. [7] present an $o(n^2)$ algorithm for the 3SUM problem on integers of bounded size. Other algorithms that break, in a similar manner, a natural quadratic barrier are for computing the edit distance between two strings and for computing the most probable sequence of states in a hidden Markov model [16, 24, 30, 33]. Another similar improvement was obtained for the all-pairs shortest path problem in [13] (see also [21] for an earlier smaller improvement for this problem).

A major difference between our algorithm and those just mentioned is that the other algorithms are designed for situations where one needs to process many discrete symbols taken from a constant size alphabet. In these situations, one can cluster together a small group of symbols into a single “super-symbol”, and process each cluster in a single step using a lookup table. In our setup the points themselves are arbitrary elements of $\mathbb{R}^2$, making our “alphabet” uncountable; thus directly grouping points and using a lookup table is inapplicable. In other words, while the other compactions are purely symbolic, we strongly exploit the underlying geometric structure of the problem to discretize the input and to obtain a clustering mechanism.

Although not detailed in this abstract, our technique can be extended (in a work in progress) so as to compute, within a comparable 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. A similar extension to the continuous Fréchet distance was employed by Buchin et al. [12]. They achieve $O(n^2 \sqrt{\log n} (\log \log n)^{3/2})$ running time on a pointer machine and $O(n^2 (\log \log n)^2)$ running time on a word RAM. It is still interesting to know whether a subquadratic running time can be achieved.

2 Overview of the Algorithm

We first study the decision problem: Given $\delta > 0$, determine whether $dF(A, B) \leq \delta$. Consider a 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 there is a directed path from $(a_0, b_0)$ to $(a_i, b_j)$ in the graph $G_M$, defined in the beginning of the introduction, and 0 otherwise. In other words, $M_{i,j}$ is 1 if the pair $(a_i, b_j)$ is reachable from the starting placement $(a_0, b_0)$ of the trip with a “leash” of length $\delta$. The goal is to compute the value of $M_{m-1,n-1}$.

The entire matrix $M$ can be computed in $O(mn)$ time in a straightforward dynamic programming technique. However, we cannot afford to compute the entire matrix $M$ explicitly if we wish to compute $M_{m-1,n-1}$ in $o(mn)$ time. Instead, we compute only certain rows of $M$. The algorithm consists of three main stages.

First, we choose a parameter $\mu = \lceil c_1 \log_2 n \rceil$ and partition the rows of $M$ into $t = \lceil m/\mu \rceil$ blocks, each of length at most $\mu + 1$, such that the first row of the $i$th block is the same as the last row of the previous block. More precisely, let $M_i$ denote the $j$th row of $M$. Then $M_i$, the $i$th block of $M$, consists of the rows $M_{i,0}, M_{i,1}, \ldots, M_{i,\mu}$, for $0 \leq i < t - 1$, and $M_{t-1}$ consists of $M_{(t-1)\mu}, \ldots, M_{m-1}$. See Figure 2. This corresponds to dividing $A$ into $t$ subchains $A_0, \ldots, A_{t-1}$ of length at most $\mu + 1$, where $A_i = (a_{i,0}, \ldots, a_{i,\mu})$, for $i < t - 1$, and $A_{t-1} = (a_{(t-1)\mu}, \ldots, a_{m-1})$. Let $\Phi_0$ and $\Phi_t$ denote the first and the last row, respectively, of $M_i$.

![Figure 2: A partition of $M$ into layers and blocks. $B$ is partitioned into subchains of length $\tau$. The automaton $M^\Phi$ reads each subchain of $B$ along with the corresponding string of $\Phi_i$, each encoded as an integer, at each step.](image)

We first describe an algorithm for processing each block $M_i$ of $M$ (Section 3). We handle $M_i$ using an approach that resembles the execution of a deterministic finite automaton $M$. We construct $M$ during preprocessing in $O(\mu^2 \mu)$ time using $O(\mu^4 \mu)$ space. Somewhat informally, the automaton is constructed from $A_i$, and we execute it on a string constructed from $\Phi_i$ and the elements of $B$. The construction and execution of $M$ exploits the geometric structure of the problem. Specifically, consider the arrangement $A_i$ of the disks of radius $\delta$ centered at the points of $A_i$. As can be seen in Figure 3, the exact location of the $B$-points is not important, but only the faces of $A_i$ in which they are contained. This allows us to construct $M$ only from the faces of $A_i$, independently of $B$. Then, assuming $\Phi_i$, the first row of $M_i$, has been computed, $M$ reads $B$ and $\Phi_i$—one point of $B$ and one bit of $\Phi_i$ at each step—and computes $\Phi_j$ in $O(n \log \log n)$ time. By choosing the constant $c_1$ in the expression for $\mu$ appropriately, processing the $i$th block takes $O(n \log \log n)$ time, so the overall time spent in processing all blocks and decid-
ing whether $dF(A, B) \leq \delta$, is $O(\frac{mn \log \log n}{\log n})$. We note that we choose $\mu = O(\log n)$ because the size of $M$ is exponential in $\mu$. In fact, as shown in Section 7, there exist point sequences $A$ and $B$ for which the size of $M$ is indeed exponential.

The above procedure already gives a subquadratic algorithm for the decision problem, but, as discussed later, this does not lead to an overall subquadratic algorithm. We next show how to improve the running time of the decision problem by another $\log n$ factor, which will then result in an overall subquadratic algorithm. This is achieved in two stages. As described in Section 4, we first transform $M$ into another automaton $M^#$ that reads $\tau = \lfloor c_2 \log n \rfloor$ points of $B$, encoded as an integer of length $O(\log n)$, and $\tau$ bits of $\Phi_1$ (again encoded as an integer of length $O(\log n)$), and outputs a sequence of $\tau$ bits of $\Phi_i$ (also encoded as an integer of length $O(\log n)$) in $O(1)$ time. That is, $M^#$ simulates $\tau$ steps of $M$ in a single step. As will be shown, the size and the construction time of $M^#$ are $2O(n \tau)$.

By choosing the constants $c_1, c_2$ appropriately, $M^#$ can be constructed in, say, $O(\sqrt{n})$ time. Ignoring the time spent in encoding $B$ and $\Phi_1$, $M^#$ computes $\Phi_1$ in $O(n \log \log n)$ time. So the total time spent in processing all the blocks, without taking into account the time spent in computing the encodings of $B$, is $O(\frac{mn \log \log n}{\log n})$.

Computing the encodings of $B$, however, requires additional work. More specifically, to process the $B$ sequence using the automaton $M$, we need to encode the faces of the arrangement $A_i$, in which the points of $B$ are contained. This requires locating the points of $B$ in $A_i$, but this takes $O(n \log \log n)$ time per block, which is too expensive. If we compute the encoding of $B$ explicitly for each block, then the overall running time for the decision procedure will remain $O(\frac{mn \log \log n}{\log n})$. Fortunately, we show (in Section 5) that the encoding of $B$ can be batched over multiple blocks and a common encoding for all these blocks can be computed once, by increasing the size of the automaton $M^#$. More precisely, we group $\lambda = \lfloor c_3 \log n \rfloor$ consecutive blocks into a single layer. The $j$th layer, denoted $L_j$, consists of the blocks $M_{j\lambda}, \ldots, M_{(j+1)\lambda-1}$; see Figure 2. We compute, in $O(n \log \log n)$ time, a single, common encoding of $B$ for all blocks in $L_j$. We modify the corresponding automata $M^# \lambda$ so that they all work with the new encoding of $B$. Assuming that the encoding of $B$ is available, the time spent in computing $\Phi_1$, the last row of block $M_i$, is still $O(\frac{mn \log \log n}{\log n})$. Hence, the total time spent on one layer, including the time spent in computing the encoding of $B$, is $O(n \log \log n)$. There are $\lfloor m / \lambda \mu \rfloor$ layers, so the overall time spent in computing $\Phi_{i-1}$, the last row $M_{i-1}$ of $M$, is $O(\frac{mn \log \log n}{\log n})$.

Putting everything together, we obtain an algorithm that solves the decision problem in $O(\frac{mn \log \log n}{\log n})$ time. Finally, to solve the optimization problem, we combine the decision procedure with a relatively simple explicit binary search, based on a significantly subquadratic procedure for distance selection [1] (Section 7). This increases the total running time by a logarithmic factor, so $dF(A, B)$ can be computed in $O(\frac{mn \log \log n}{\log n})$ time.

Figure 3: 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 $\delta$) centered at the points of $A$ form the arrangement $A$. The stepping stones of the $B$-frog are the hollow points. In this example, a legal path for the two frogs is $((a_0, b_0), (a_1, b_0), (a_1, b_1), (a_1, b_2), (a_2, b_2), (a_3, b_2))$.

3 Handling a Single Block

In this section, we describe the algorithm for processing a single block $M$ of the matrix $M$. Let $A$ be the subchain of $A$ corresponding to $M$: if $M$ is the $i$th block overall, then $A = (a_{\mu}, \ldots, a_{(i+1)\mu})$. For simplicity, we use $(a_0, \ldots, a_\mu)$ to denote the sequence of vertices in $A$, and $\Phi = \varphi_0, \ldots, \varphi_{n-1}$ (resp., $\bar{\Phi} = \bar{\varphi}_0, \ldots, \bar{\varphi}_{n-1}$) to denote the first (resp., last) row of $M$. That is, $\Phi$ (resp., $\bar{\Phi}$) is the row of $M$ (or $M$) corresponding to the starting point $a_0$ of $A$ (resp., the terminal point $a_\mu$). If $M$ is the first block of $M$, then

$$\varphi_i = \begin{cases} 1 & \text{if } \|a_0 - b_j\| \leq \delta \text{ for all } j \leq i, \\ 0 & \text{otherwise.} \end{cases}$$

If $M$ is not the first block, then $\Phi$ is computed as the output sequence $\Phi$ from the processing of the previous block. In either case, given $\Phi$ and $A$, the goal is to compute $\bar{\Phi}$.

Let $G = G_A(A, B)$ be the subgraph of $G_A(A, B)$ induced by $A \times B$. Recall that $\bar{\varphi}_j = 1$ if there is a path in $G_A(A, B)$ from $(a_0, b_0)$ to $(a_\mu, b_j)$. Such a path contains a vertex $(a_k, b_k)$ for some $k \leq j$. Since $(a_0, b_k)$ is reachable from $(a_0, b_0)$ in $G_A(A, B)$, we have, by definition, $\varphi_k = 1$. Furthermore, $(a_\mu, b_j)$ is reachable
from \((a_0, b_k)\) in \(G\); see Figure 4. Conversely, if, for some \(k\), \((a_\mu, b_j)\) is reachable from \((a_0, b_k)\) in \(G\) and \(\varphi_k = 1\) then \((a_\mu, b_j)\) is reachable from \((a_0, b_0)\). Hence, we conclude that

\[
\hat{\varphi}_j = 1 \text{ iff } \exists k \leq j \text{ such that } \varphi_k = 1 \text{ and } \exists \text{ a path from } (a_0, b_k) \text{ to } (a_\mu, b_j) \text{ in } G.
\]

In general, we call a pair \((a_\mu, b_j)\) is \(\in A \times B\) accessible if there exists \(k \leq j\) such that \(\varphi_k = 1\) and there is a path from \((a_0, b_k)\) to \((a_\mu, b_j)\) in \(G\).

We are thus interested in computing all accessible pairs in \(\{a_\mu\} \times B\). Instead of computing the graph \(G\) explicitly and searching in \(G\) (steps that are too expensive for our goal), we make the following observation. For any \(0 \leq j < n\), we define \(X_j \subseteq A\) as

\[
X_j = \{a \in A \mid (a, b_j) \text{ is accessible}\}.
\]

Obviously, \(\hat{\varphi}_j = 1\) if \(a_\mu \in X_j\). We show below how to preprocess \(A\) and construct a finite automaton \(M\) that, given \(X_j, b_j, b_{j+1}\), and \(\varphi_{j+1}\), can compute \(X_{j+1}\) in \(O(\log \log n)\) time. The actual automaton that we will eventually use will be more contrived and more efficient, but the description of \(M\) will help to present the main ideas of the technique in a simpler manner.

![Figure 4: Passing through the block M from \((a_0, b_0)\) to \((a_\mu, b_j)\). This path is meaningful only if its starting point \((a_0, b_0)\) is reachable from \((a_0, b_0)\), i.e., \(\varphi_k = 1\).](image)

Before describing the construction of \(M\), we show how to replace the sequence \(B\), which is a sequence of points in \(\mathbb{R}^2\), with a sequence over a finite (albeit not constant) alphabet. For \(0 \leq j < \mu\), let \(D_j\) be the disk of radius \(\delta\) centered at \(a_j\). Set \(D = \{D_j \mid 0 \leq j < \mu\}\). Construct the arrangement \(A = \mathcal{A}(D)\) of the disks and preprocess it for fast point-location queries using any standard data structure [17]. Associate with each face \(f\) (of any dimension) of \(A\) the subset \(\mathcal{D}_f \subseteq D\) of disks that contain \(f\). Set \(A_f = \{a_j \mid D_j \in \mathcal{D}_f\}\). For any point \(p \in f\), \(A_f\) is the subset of points of \(A\) that are within distance \(\delta\) from \(p\). Let \(F\) be the set of all faces (of any dimension) in \(A\); \(|F| = O(\mu^2)\). For a point \(p \in \mathbb{R}^2\), let \(\psi(p)\) denote the face of \(A\) that contains \(p\). For \(j < n\), let \(f_j = \psi(b_j)\). We observe that the graph \(G\) does not depend on the exact coordinates of the points in \(B\) but only on the faces of \(A\) that contain the points of \(B\). More precisely, for any sequence \(B' = (b_0, \ldots, b_{n-1})\) of points such that \(\psi(b_j') = f_j\) for all \(j \leq n\), \(\exists g(A, B')\) is the same as \(G\). Hence, we will replace \(B\) with the face sequence \(F = (f_0, \ldots, f_{n-1})\). By performing a point-location query in \(A\) with each \(b_j \in B\), \(F\) can be computed in \(O(n \log \log n)\) time.

We now describe the automaton \(M = (K, \Sigma, \sigma, \Delta)\), where \(K\) is the set of its states, \(\Sigma\) its alphabet, \(s\) its initial state, and \(\Delta\) its transition function. First, \(\Sigma = \mathcal{F} \times \{0, 1\}\), i.e., at each step \(M\) reads one symbol from the sequence \(F\) and one bit from \(\Phi\); clearly, \(|\Sigma| = O(\mu^2)\). A state in \(K\) is represented as a pair \((f, S_f)\) in \(\mathcal{F} \times 2^\mathcal{F}\), which has the following interpretation: After reading a prefix \(B_k\) of \(B\), which ends at a point \(b_k\) of \(B\), if \(M\) enters the state \((f, S_f)\) then (i) \(b_k \in f\), (ii) \(S_f \subseteq A_f\), and (iii) \(\mathcal{F} \times \{b_k\}\) is the set of all accessible pairs in the column \(A \times \{b_k\}\) of \(M\), i.e., \(X_k = S_f\). In other words, \(\alpha \in S_f\) if and only if there exists a point \(b_k \in B_k\) such that \(\varphi_l = 1\) and there is a path in \(G\) from \((a_0, b_0)\) to \((a_\mu, b_k)\). The set \(S_f\) has the property, dictated by the edges in \(G\), that if \(a_j \in S_f\) and \(a_{j+1} \in A_f\) then \(a_{j+1}\) also belongs to \(S_f\). The initial state \(s\) is set to \((f, \emptyset)\) for some arbitrary face \(f \in \mathcal{F}\). Note that \(|K| = O(\mu^2 2^\mu)\).

Next, we define the transition function \(\Delta : K \times \Sigma \rightarrow K \times \{0, 1\}\). For a pair \((q, \sigma) \in K \times \Sigma\), \(\Delta(q, \sigma) = (q', \varphi')\) means that when \(M\) reads a symbol \(\sigma \in \Sigma\) while in state \(q\), it outputs the bit \(\varphi\) and enters the state \(q'\). Specifically, if \(q = (f, S_f)\) and \(\sigma = (g, \varphi)\) then \(\Delta(q, \sigma) = ((g, S_g), \varphi')\) when the following holds:

(i) A point \(a_k \in A_g\) belongs to \(S_g\) if either \(a_k \in S_f\) or there exists \(j \leq k\) such that \(a_j \in S_f\) and the entire run \((a_{j+1}, \ldots, a_k)\) is contained in \(A_g\).

(ii) In addition, \(a_k \in S_g\) if \(\varphi = 1\) and \((a_0, \ldots, a_k)\) is \(\in S_g\).

(iii) \(\varphi = 1\) if \(a_\mu \in S_g\) and 0 otherwise.

![Figure 5: A transition from a state \((f_0, S_{f_0})\) to a state \((f_1, S_{f_1})\). Here \(S_{f_0}\) contains \(a_0\) and \(a_1\), and when moving to state \((f_1, S_{f_1})\) (the \(-\)frog jumps from \(f_0\) to \(f_1\)), \(a_0\) is no longer contained in \(S_{f_1}\).](image)
The transition from the state \((f, S_f)\) to \((g, S_g)\) after reading a point \(b_i \in B\) and \(\varphi_r \in \Phi\) (note that \(b_{r-1} \in f\) and \(b_r \in g\)) means that at least one of the following conditions holds for a point \(\alpha_k \in S_g\):

(i) The pair \((\alpha_k, b_{r-1})\) is accessible and \((\alpha_k, b_r)\) can be reached from it by the edge \(((\alpha_k, b_{r-1}), (\alpha_k, b_r))\) of \(G\).

(ii) There exists \(j \leq k\) such that the pair \((\alpha_j, b_{r-1})\) is accessible and \((\alpha_k, b_r)\) can be reached from it by a path consisting of the edge \(((\alpha_j, b_{r-1}), (\alpha_{j+1}, b_r))\) followed by the edges \(((\alpha_{j+1}, b_r), (\alpha_{j+2}, b_r)), \ldots, ((\alpha_{k-1}, b_r), (\alpha_k, b_r))\).

(iii) If \(\varphi_r = 1\), the pair \((\alpha_0, b_r)\) is accessible and \((\alpha_k, b_r)\) can be reached from it by the edges \(((\alpha_0, b_r), (\alpha_1, b_r)), \ldots, ((\alpha_{k-1}, b_r), (\alpha_k, b_r))\).

Consider for example the transition from a state \((f, S_f)\) to a state \((g, S_g)\) when \(f = g\). If \(\varphi_r = 0\) then \(S_f = S_g\). However, if \(\varphi_r = 1\) then \(S_f\) might be different from \(S_g\). In this case, \(S_g\) contains the points of \(S_f\), and the maximal prefix of points in \(A\) that is contained in \(A_g\). If \(\varphi_{r-1} = 0\) this prefix may contain new points.

The transition function \(\Delta\) can be stored as a look-up table, called the transition table, of size \(|K| \times |\Sigma| = O(\mu^2\tau^2)\), which is indexed by \(f, S_f, g, \) and the bit \(\varphi\). Furthermore, each entry of \(\Delta\) can be computed in \(O(\mu)\) time in a straightforward manner. Hence, \(M\) can be constructed in \(O(\mu^2\tau^2)\) time. By choosing the constant \(c_1\) in \(\mu\) appropriately, we can ensure that \(M\) can be constructed in \(O(n^{1/4})\) time, say.

After having constructed \(M\), we run it on the face sequence \(F\) and the first row \(\Phi\) of \(M\) as follows. Initially, \(M\) is in the state \(s\). Suppose \(M\) is in state \((f_j, S_{f_j})\) after reading \((f_0, \varphi_0), \ldots, (f_j, \varphi_j)\). In the next step, it reads \((f_{j+1}, \varphi_{j+1})\). Suppose \(\Delta((f_j, S_{f_j}), (f_{j+1}, \varphi_{j+1})) = ((f_{j+1}, S_{f_{j+1}}), \tilde{\varphi})\). Then \(M\) sets \(\tilde{\varphi}_{j+1} := \tilde{\varphi}\) and enters the state \((f_{j+1}, S_{f_{j+1}})\). (Note that the first move, from \(s\), can be triggered only by a transition corresponding to condition (iii), and not by crossing from the face of \(s\).

With an appropriate indexing of the states, and using the transition table representation of \(\Delta\), each step of \(M\) takes \(O(1)\) time. Hence, after computing the face sequence \(F\) and constructing \(M\), \(\Phi\) can be produced in \(O(n)\) time. Putting everything together, \(M\) can be processed in \(O(n \log n)\) time using \(O(n)\) space.

Repeating this procedure for all \(t\) blocks of \(M\) and reporting \(YES\) if \(M_{n-1,n-1} = 1\) and \(NO\) otherwise, we obtain the following.

**Lemma 3.1.** Let \(A\) and \(B\) be two sequences of points of size \(m\) and \(n\), respectively, in \(\mathbb{R}^2\), with \(m \leq n\), and let \(\delta > 0\) be a parameter. The decision problem, whether \(d_F(A, B) \leq \delta\), can be solved in \(O(mn \log \log n)\) time using \(O(n)\) space.

**Remark.** We choose \(\mu = O(\log n)\) because the size of \(M\) is exponential in \(\mu\). As shown in Section 7, there exists a point sequence \(A\) for which the size of \(M\) is indeed exponential, in the sense that there exist exponentially many point sequences \(B\) that lead to these exponentially many states. This lower bound, however, does not preclude the possibility of constructing a more clever automaton, by exploiting further properties of the discrete Frechet distance (e.g., the two frogs cannot backtrack while traversing the two sets of stones), whose size is polynomial in \(\mu\). The existence, and efficient construction, of such an automaton has so far remained elusive for us.

### 4 Handling Subchains of \(B\) in a Single Step

In this section we describe a different automaton for processing a block \(M\) that computes \(\Phi\) faster, provided the face sequence \(F\) as well as the first row \(\Phi\) are given in a “compressed” form. We will follow the notation from the previous section.

Let \(\tau = \lceil \frac{\log \log n}{c_2} \rceil\) be a parameter, where \(c_2\) is a constant whose value will be chosen later. For simplicity, we assume that \(n\) is divisible by \(\tau\). We transform the automaton \(M\), described in the previous section for processing a block \(M\), into another automaton \(M^\#\) that, in each step, reads a string of \(\tau\) consecutive faces of the face sequence \(F\) and the corresponding string of \(\tau\) bits of \(\Phi\), and outputs \(\tau\) bits of \(\Phi\). Namely, we partition \(F\) into \(u = \lceil n/\tau \rceil\) substrings \(F_1, \ldots, F_u\) of size \(\tau\) each and encode each \(F_i\) as a \(\beta\)-bit integer \(e(F_i)\), for \(\beta = O(\log n)\), as described below. We also partition \(\Phi\) into \(u\) substrings \(\varphi_1, \ldots, \varphi_u\) and view each \(\varphi_r\) as a \(\tau\)-bit integer \(e(\varphi_r)\). Similarly, let \(\varphi_1, \ldots, \varphi_u\) be the partition of \(\Phi\) into \(u\) strings of length \(\tau\) each, and treat each \(\varphi_r\) as an integer \(e(\varphi_r)\). \(M^\#\) will compute \(e(\varphi_1), \ldots, e(\varphi_u)\).

If \(M\) is the first block, then \(e(\varphi_1) \ldots e(\varphi_u)\) can be computed in \(O(n)\) time; otherwise they will be the output of the automaton that processed the previous block. Since \(e(F_1)\) and \(e(\varphi_1)\) are \(O(\log n)\)-bit integers, each of them can be stored using \(O(1)\) space and can be read/written in \(O(1)\) time. In each step, the new automaton \(M^\#\) will read \(e(F_1)\), \(e(\varphi_1)\) and output \(e(\varphi_1)\) in \(O(1)\) time.

We now describe the structure of \(M^\# = (K, \Sigma^\#, s, \Delta^\#)\), which like \(M\) is constructed independently of \(F\) and \(\Phi\). \(M^\#\) has the same set of states as \(M\), and its initial state is the same as that of \(M\). \(\Sigma^\# = \{0 : 2^{2\beta-1} \times [0 : 2^{\beta-1}]\}, i.e., each symbol in \(\Sigma^\#\) is a pair of integers of length \(\beta\) and \(\tau\), respectively, intended to encode a pair \((F, \varphi)\), where, as above, \(F\) is a sequence of \(\tau\) faces and \(\varphi\) is a string of \(\tau\) bits. Let
\((f,S_f)\) be a state of \(M, \Xi = \langle \xi_1, \ldots, \xi_r \rangle\) a sequence of \(\tau\) faces of \(\mathcal{F}\), and \(b = (b_1, \ldots, b_r)\) a sequence of \(\tau\) bits. Suppose we run \(M\) on \(\Xi\) and \(b\) starting at the state \((f,S_f)\). If \(M\) outputs the string \(\tilde{b} = (b_1, \ldots, b_r)\) and enters the state \((\xi_r, S_r)\) after processing the sequence of \(\tau\) matching pairs from \(\Xi\) and \(b\), then we set \(\Delta^\#((f,S_f),(e(\Xi),e(\tilde{b}))) = ((\xi_r, S_r),e(\tilde{b}))\).

Since an output character of \(M^\#\) depends on a sequence of states of \(M\), we associate an output value with each transition edge of \(M^\#\) (in other words, we construct \(M^\#\) as a Mealy machine.) Let \(\mathcal{F}'\) denote the family of all sequences of faces in \(\mathcal{F}\) of length \(\tau\). To construct \(M^\#\), we first construct \(M\). Then, for each state \((f,S_f)\) \(\in M\), for each \(\tau\)-length face sequence \(\Xi \in \mathcal{F}'\), and for each \(\tau\)-bit binary string \(b\) we compute \(\Delta^\#((f,S_f),(e(\Xi),e(b)))\) by running \(M\) on \(\Xi\) and \(b\) starting from the state \((f,S_f)\) as just described. We store \(\Delta^\#\) as a look-up table of size \(O(\mu^22^\nu) \times 2^{2\beta+\tau} = O(\mu^22^\nu+\beta+\tau)\). We recall that \(\beta = O(\log n)\) is the length of the details of the encoding of a \(\tau\)-long face sequence. After providing the details of the encoding shortly, it will become clear that, as promised, \(\beta = \Theta(\log n)\). Each entry of \(\Delta^\#\) can be computed in \(O(\tau)\) time (after having constructed \(M\)). By choosing the constant \(c_2\) appropriately, the total construction time and size of \(\Delta^\#\) is \(O(\sqrt{n})\), say.

To complete the description of \(M^\#\), we now describe a simple encoding scheme that converts a \(\tau\)-long string \(\Xi = \langle \xi_1, \ldots, \xi_r \rangle \in \mathcal{F}'\) to a \(\beta\)-bit long integer. Since \(\mathcal{F}' = O(\mu^2)\), each face \(f \in \mathcal{F}\) can be labeled as an integer \(e(f)\) in the range \([0,|\mathcal{F}|-1]\), using \(\nu = \lfloor \log_2(|\mathcal{F}|) \rfloor = O(\log \mu)\) bits. For an integer \(x\), let \((x)\) denote its binary representation, then

\begin{equation}
    e(\Xi) = \sum_{i=1}^{r} e(\xi_i) \cdot 2^{\nu(i-1)}.
\end{equation}

That is, \((e(\Xi)) = (e(\xi_1)) \circ \cdots \circ (e(\xi_r))\). Note that the length of \(e(\Xi)\) is

\[\beta = O(\tau \nu) = O(\tau \log \mu) = O\left(\frac{\log n}{\log \log n} \cdot \log \log n\right) = O(\log n),\]

as stated above, and that \(e(\Xi)\) can be computed in \(O(\tau)\) time. Set

\[e(F) = (e(F_1), \ldots, e(F_u))\]

and

\[e(\Phi) = (e(\varphi_1), \ldots, e(\varphi_u)).\]

Given \(M^\#, e(F)\), and \(e(\Phi)\), we can compute \(e(\Phi)\) (defined analogously to \(e(\Phi)\)) by running \(M^\#\) on \(e(F)\) and \(e(\Phi)\). Suppose \(M^\#\) is in state \(q_{i-1} \in K\) after reading the first \(i - 1\) integer pairs of matching entries of \(e(F)\) and \(e(\Phi)\). In the next step, \(M^\#\) reads the pair of integers \((e(F_i),e(\varphi_i))\). If \(\Delta^\#(q_{i-1},(e(F_i),e(\varphi_i))) = (q_i,b)\), then \(M^\#\) outputs the integer \(b\) and enters the state \(q_i\). The \(i\)th step takes \(O(1)\) time, because all it needs to do is to access and fetch the entry of the look-up table at the pair of (already available) indices \((e(F_i),e(\varphi_i))\). Hence, after having constructed \(M^\#\) and the encoding \(e(F)\), \(e(\Phi)\) can be computed in \(O(\nu) = O\left(\frac{n \log \log n}{\log n}\right)\) time. Since \(M^\#\) can be constructed in \(O(\sqrt{n})\) time, we obtain the following.

\textbf{Lemma 4.1.} Let \(M\) be a block of \(\mathcal{M}\) and \(\Lambda\) the subchain corresponding to \(M\). Given \(e(F)\), the encoding of the face sequence corresponding to \(B\), and \(e(\Phi)\), the encoding of the first row of \(M\), the encoding \(e(\Phi)\) of the last row can be computed in \(O\left(\frac{n \log \log n}{\log n}\right)\) time using \(O(n)\) space.

\section{Handling a Layer}

Lemma 4.1 already provides the bound that we are after, but it relies on the availability of the encoding of \(B\) that it needs to use. In this section we describe how to compute this encodings of \(B\) more efficiently by batching it over multiple blocks. This, combined with the tools developed above, will finally lead to an \(O(n \min \log \log n)\) algorithm for the decision problem.

Put \(\lambda = [c_3 \log n]\), where \(c_3\) is a constant to be chosen later. We partition the blocks \(M_0, \ldots, M_{c_3}\) into \(\ell = \lceil \frac{\lambda}{\lambda} \rceil\) layers, \(L_0, \ldots, L_{\ell-1}\), where \(L_i\) contains the blocks \(M_{i\lambda}, \ldots, M_{(i+1)\lambda-1}\). We describe the procedure for the layer \(L_0\); the same procedure works for other layers.

We first describe how to compute a single encoding of \(B\) for all blocks in \(L_0\), and then describe the construction of the automaton for each block in \(L_0\).

\textbf{Encoding of \(B\).} For simplicity, we use \(L\) to denote the layer \(L_0\). \(L\) contains the blocks \(M_0, \ldots, M_{\lambda-1}\). Let \(\Lambda = \langle a_0, \ldots, a_{\lambda\mu} \rangle\) be the subchain of \(\mathcal{A}\) spanned by the blocks in \(L\); \(|\Lambda| = O(\mu \lambda) = O(\log^2 n)\). We construct the arrangement \(\mathcal{A}\) of the disks of radius \(\delta\) centered at the points of \(\Lambda\). (That is, we construct one arrangement for the entire layer instead of a separate arrangement for each of its blocks.) Let \(\mathcal{F}\) denote the set of faces in \(\mathcal{A}\); \(|\mathcal{F}| = O(\mu^2 \lambda^2) \leq c \log n^4 n\) for some constant \(c\) that depends on \(c_1\) and \(c_3\). We preprocess \(\mathcal{A}\), in \(O(\log^4 n \log \log n)\) time, for answering point-location queries, using any of the standard techniques \([17]\); each query takes \(O(\log \log n)\) time. Fix a block \(M_j\), for some 0 \(\leq j < \lambda\). Let \(A_j\) be the arrangement constructed only on the disks centered at the points of \(A_j\). Each face \(f\) of \(\mathcal{A}\) is a subface of a face \(f^{(j)}\) of \(A_j\) (see Figure 6). We find these correspondences by preprocessing each of the arrangements \(A_i\) for fast point location. Then, for each face \(f\) of \(\mathcal{A}\) we pick an arbitrary point in \(f\) and locate
it in each \( A_j \). For each \( A_j \) we obtain the coarser face \( f^{(i)} \), and we store a pointer from \( f^{(i)} \) to \( f \). Thus, each “super-face” \( f^{(i)} \) store pointers to the faces of \( \hat{A} \) that lie inside \( f^{(i)} \). Let \( F_y \) denote the set of faces in \( \hat{A} \) that lie inside a face \( g \).

![Diagram of \( \hat{A} \)](image)

Next, for each point \( b_i \) of the B-sequence, we locate the face \( f_i \) of \( \hat{A} \) containing \( b_i \), using the point-location structure. This takes \( O(n \log n) \) time (now per layer and not per block). We obtain a sequence \( F = (f_0, f_1, \ldots, f_{n-1}) \) of faces of \( \hat{A} \), and we partition it into \( u \) subsequences \( F_1, \ldots, F_u \), each consisting of \( \tau \) consecutive faces, where \( \tau = \lfloor c_2 \log n / \log \log n \rfloor \) and \( u = \lceil n / \tau \rceil = \Theta(n \log \log n / \log n) \), as above.

We encode \( F \) in a manner similar to the previous section, except that we use the faces in \( F \) instead of those in the coarser block subarrangements. Since \( |F| \leq c \log^4 n \), each face \( f \) in \( F \) can be labeled as an integer \( \hat{e}(f) \in [0 : c \log^4 n - 1] \) with \( \hat{e} = \lfloor \log_2 |F| \rfloor = O(\log \log n) \) bits. (Note that \( \hat{e} \) is roughly twice as large as \( \nu \), the length of the encoding of a face in the previous section.) For each subsequence \( F_k = (f_{(k-1)\tau}, \ldots, f_{k\tau-1}) \) (\( 1 \leq k \leq u \)), we compute its encoding

\[
\hat{e}(F_k) = \sum_{i=0}^{\tau-1} \hat{e}(f_{(k-1)\tau+i}) \cdot 2^{\hat{e}i}.
\]

Let \( \beta \) denote the length of the binary string \( \langle \hat{e}(F_k) \rangle \). Note that \( \beta = \tau \hat{e} = O(\log n) \), and thus \( \hat{e}(F_k) \) can be stored in \( O(1) \) space and accessed in \( O(1) \) time. As earlier, \( \hat{e}(F_k) \) can be computed in \( O(\tau) \) time. We set

\[
\hat{e}(F) = \hat{e}(F_1), \ldots, \hat{e}(F_u),
\]

which is the common encoding of \( F \) for all blocks in the layer \( L \). Given \( B \), the overall time spent in computing \( \hat{e}(F) \) is \( O(n \log \log n) \).

We stress again that \( \hat{e}(F) \) is computed only once within the layer \( L \). We also remark that the encoding \( e(\Phi_j) \) of \( \Phi_j \) for each block \( M_j \) in \( L \) remains the same as in Section 4 and is computed explicitly for each block, as described below.

**Processing a block in \( L \).** Suppose we have processed the blocks \( M_0, \ldots, M_{i-1} \) of \( L \) and computed \( e(\Phi_0), \ldots, e(\Phi_{i-1}) \). To process \( M_i \), we preprocess it into an automaton, independent of \( B \) and \( \Phi_i = \Phi_{i-1} \), and then use it to compute \( e(\Phi_i) \). Let \( M_i, M_i^\# \) denote the automata constructed for the block \( M_i \) in Sections 3 and 4, respectively. We first describe an “unpacked” automaton \( L_i \), analogous of \( M_i \), that, in each step, reads one face of \( F \) and one bit of \( \Phi \) and outputs a bit of \( \Phi_i \). We then transform \( L_i \) into another compacted automaton \( L_i^\# \) that reads \( \hat{e}(F) \) and \( e(\Phi_i) \), the encoded (compressed) sequences of \( F \) and \( \Phi_i \), respectively, as in Section 4. In a sense, we repeat the description of the construction of \( M_i^\# \) from \( M_i \), but we also note some technical issues that require a different treatment, adapt the various parameters to the new setup, and make sure that the resulting performance satisfies the desired bounds.

The automaton \( L_i = (K_i, \Sigma_i, \hat{e}, \Delta_i) \) is very similar to \( M_i \) except that it is defined over the faces in \( F \) (instead of the faces of the individual arrangements \( A_i \)). Namely, \( \Sigma_i = F \times \{0, 1\} \), and each state in \( K_i \) is a pair \((f, \varphi)\) in \( F \times 2^\nu \), with the same interpretation as for \( M_i \); \( |\Sigma_i| = O(\mu^3 \lambda^2) \) and \( |K_i| = O(\mu^3 \lambda^2 2^\nu) \) (we use more faces but still consider only the points of \( A_i \)).

The transition function \( \Delta_i : K_i \times \Sigma_i \to K_i \times \{0, 1\} \) is also defined analogously. Since each face \( f \in F \) is a subface of a face \( f^{(i)} \in A_i \), \( L_i \) is a refinement of \( M_i = (K_i, \Sigma_i, \hat{e}, \Delta_i) \) in the sense that \( M_i \) can be obtained from \( L_i \) by partitioning the states of \( L_i \) into equivalence classes and compressing the states in each equivalence class into a single state. More precisely, for a face \( f \in F \) lying in a face \( f^{(i)} \in A_i \), there is a state \((f, \varphi) \in K_i \) if and only if \((f^{(i)}, \varphi) \in K_{i-1} \). Similarly, for two faces \( f, g \in F \) lying in the faces of \( f^{(i)} \) and \( g^{(i)} \) of \( A_i \) respectively, \( (f, \varphi)(g, \varphi) = ((f, \varphi) \cup (g, \varphi)) \) if and only if \( (f^{(i)}, \varphi)(g^{(i)}, \varphi) = ((f^{(i)}, \varphi) \cup (g^{(i)}, \varphi)) \).

We thus construct \( L_i \) from \( M_i \) as follows. For each state \((f, S_f) \in K_i \), we follow the pointers from \( f \) to the faces of \( A \) contained in \( f \), thereby obtaining the set \( F_f \). Then, for every \( f \in F_f \), we add the pair \((f, S_f)\) to \( K_i \). Similarly, for each entry \( \Delta_i((f, S_f), (g, \varphi)) = ((g, S_g), \varphi) \) in the transition table of \( M_i \), we obtain the set \( F_{g} \) by following the pointers from \( g \) to the faces of \( A \) contained in \( g \). Then, we add the entry \( \Delta_i((f, S_f), (g, \varphi)) = ((g, S_g), \varphi) \), for every \( f \in F_f \) and every \( g \in F_{g} \), to the transition table of \( L_i \). The size of \( \Delta_i \) is \(|K_i| \cdot |\Sigma_i| = O(\mu^4 \lambda^2 2^\nu) \), and the construction time is proportional to this bound.

Next, we construct the automaton \( L_i^\# \) from \( L_i \) in the same manner as in Section 4. That is, when \( L_i^\# \)
is at state \((f, S_t)\) and reads a pair of integers \(e(F_k)\) and \(e(\varphi_k)\), where \(F_k\) is a \(\tau\)-long sequence of faces in \(F\) and \(\varphi_k\) is a \(\tau\)-bit binary string, it moves to the state \((f, S_{t+1})\), where \(f_t\) is the last face of \(F_k\), and outputs the integer \(e(\varphi_k)\), obtained by running the new \(L_u\) on \(F_k\) and \(\varphi_k\). The alphabet of \(L_u\) consists of pairs of integers of length \(\beta\) and \(\tau\) and \(L_u\) has \(|K_1|\) states, so its size is \(O(\mu^2 \lambda^2 \mu^2) \times 2^{3\beta + \tau} = O(\mu^2 \lambda^2 \mu^{2 + \beta + \tau})\). Moreover, each entry of \(\Delta_t\) can be computed in \(O(\tau)\) time. By choosing the constants \(c_1, c_2, c_3\) appropriately, as in the previous section, \(L_u\) can be constructed in \(O(n \log \log n / \log n)\) time.

Finally, we run \(L_u\) on the encoded sequences \(e(\tilde{F}) = e(F_1), \ldots, e(F_n)\) and \(e(\tilde{F}_i) = e(\varphi_1), \ldots, e(\varphi_n)\) to generate the encoding \(e(\tilde{F}_i) = e(\varphi_1), \ldots, e(\varphi_n)\) of \(\tilde{F}_i\), the compacted representation of the last row of the block \(M_i\). In the \(j\)th step, \(L_u\) reads \(e(F_j), e(\varphi_j)\) and outputs \(e(\varphi_j)\), in \(O(1)\) time. The total time spent in this phase is \(O(u) = O(n \log \log n / \log n)\).

Putting pieces together. After constructing the encoding of \(B\) for a layer \(L_i\), in \(O(n \log \log n)\) time, all blocks of \(L_i\) can be processed in a total time of \(O(n \log \log n)\). Hence, processing \(L_i\) takes \(O(n \log \log n)\) time and \(O(n)\) space. Since there are \([m/\mu]\) layers, the total time spent in processing all of them, and in computing the last row \(M_{m-1}\) of \(M\), is \(O(m \log \log n / \log n)\).

We thus obtain the following intermediate result.

Theorem 5.1. Let \(A, B\) be two sequences of points in \(\mathbb{R}^2\) of sizes \(m\) and \(n\), respectively, with \(m \leq n\), and let \(\delta > 0\) be a parameter. Then the decision problem, whether \(dF(A, B) \leq \delta\), can be solved in \(O(mn \log \log n / \log^2 n)\) time using \(O(n)\) space.

6 The Optimization Procedure

We use the decision procedure described above to solve the optimization problem, as follows. First note that the critical values of \(\delta\), in which an edge is added to the graph \(\mathcal{G}_\delta\) (as \(\delta\) 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 \([m/n^{1/2}]\) 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 \(\delta\) centered at the points of \(A_i\), and locate the points of \(B\) in this arrangement, exactly as in [1].

Altogether, this can easily produce the number of pairs in \(A \times B\) at distance at most \(\delta\), which is what the decision procedure needs. The overall cost of this procedure is \(O(n^{3/2} \log n)\). Finally, omitting the fairly routine details, we solve the optimization version of the distance selection algorithm using parametric searching, increasing the running time to \(O(n^{3/2} \log^4 n)\). This running time is subsumed by the cost of the decision procedure of the preceding sections.

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

Theorem 6.1. Let \(A, B\) be two sequences of points in \(\mathbb{R}^2\) of sizes \(m\) and \(n\), respectively, with \(m \leq n\). Then the discrete Fréchet distance between \(A\) and \(B\) can be computed in \(O((mn \log \log n) / \log n)\) time using \(O(n)\) space.

7 A Lower Bound on the Size of the Automaton

An interesting question that pops up right away in the design of the algorithm is how large can the automaton be. That is, how many states (and transition rules) can one have. Unfortunately, we give here a construction for the sequences \(A\) and \(B\) that shows that the automaton should have a number of states which is exponential in the size of \(A\) in order to be prepared for any sequence \(B\). This is the reason for using blocks of size \(\Theta(\log n)\), with a sufficiently small constant of proportionality, because using blocks of larger size may cause our respective automata to be too large for a subquadratic algorithm.

The construction, depicted in Figure 7, uses an even number of disks; with a slight abuse of notation, we denote their number by \(2m\). Enumerate the disks as \(D_0, D_1, \ldots, D_{2m-1}\) and their respective centers as \(a_0, a_1, \ldots, a_{2m-1}\). All these centers lie on the \(x\)-axis in the right-to-left order \(a_0, a_2, a_4, \ldots, a_{2m-2}, a_1, a_3, \ldots, a_{2m-1}\). The centers of the odd-indexed disks (red disks for short) are sufficiently close to each other, so that these disks have a large common intersection. The even-indexed disks (blue disks for short) are placed so that, for each \(k = 0, \ldots, m-1\),

Although there are more efficient algorithms for distance selection, which run in close to \(O(n^{4/3})\) time [1, 26], this simple-minded solution (more than) suffices for our purpose, and it has the advantage that it only uses linear storage.
$D_{2k}$ intersects $D_{2k+1}$ (in a small cap) but is disjoint from $D_{2k+3}$ (the second condition is vacuous for $k = m - 1$).

Figure 7: A configuration of disks with an exponential number of states. The red disks are drawn solid and the blue disks are drawn dashed. The faces $f_0, f_0', \ldots, f_{m-1}$ are highlighted.

We next place $2m + 1$ points $b_0, b_0', b_1, b'_1, \ldots, b_m, b'_m$ (or, rather, select $2m + 1$ corresponding faces $f_0, f'_0, f_1, f'_1, \ldots, f_{m-1}, f'_{m-1}, f_m$ of the resulting arrangement of the $2m$ disks). For each $i = 0, \ldots, m - 1$, we take $f_i$ to be the cap $D_{2i} \cap D_{2i+1}$. By construction, and as shown in the figure, these are indeed faces of the arrangement. We take $f'_j$ to be the face lying directly above $f_j$, so that in order to go from $f_j$ to $f'_j$ we need to exit the two disks $D_{2j}$ and $D_{2j+1}$ (and do not have to cross the boundary of any other disk). Finally, we take $f_m$ to be the intersection face of all the red (odd-indexed) disks.

We regard $(a_0, b_0)$ as the starting position of the frogs, where $b_0$ is any point in $f_0$ and $a_0$ is the center of $D_0$, and the goal position is $(a_{2m-1}, b_m)$, where $b_m$ is any point in $f_m$ and $a_{2m-1}$ is the center of $D_{2m-1}$.

By construction, $D_{f_m}$ consists of all the $m$ red disks. For a face $f_i$, we define $A_f$ analogously to our definition for a block in Section 3. We claim that for every subset $S \subseteq A_{f_m}$, the automaton can reach the state $(f_m, S)$, 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 $b_0$ and ends at $b_m$, (ii) contains all the points $b_0, b_1, \ldots, b_m$ (in this order), and (iii) contains some of the points $b'_0, \ldots, b'_{m-1}$, 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 point $a_i \in S$, as the $B$-frog moves through the sequence $B_S$, the $A$-frog can execute a sequence of corresponding moves, so that it reaches $a_i$ at the end, and this cannot be achieved (for the same sequence $B_S$) for any $a_i \notin S$. For simplicity, we only specify the sequence of faces of $A$ containing the points of $B$, rather than the points themselves (although the figure depicts the points too).

So let $S \subseteq A_{f_m}$ be given. We associate with $S$ the following sequence $F_S$ of faces. We start with the subsequence $(f_0, f_1, \ldots, f_{m-1}, f_m)$ and, for each point $a_{2k+1} \notin S$, 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 odd-indexed points of $A$ not in $S$) and of short horizontal moves (for odd-indexed $A$-points in $S$).

We next argue that if the automaton processes the sequence $F_S$ it indeed reaches the state $(f_m, S)$ eventually. Consider an odd-indexed point $a_{2k+1} \in A$ not in $S$. When the $B$-frog follows the detour from $f_k$ to $f'_k$ and then to $f_{k+1}$, it leaves $D_{2k}$ and $D_{2k+1}$ and then re-enters $D_{2k+1}$ (and $D_{2k+3}$). The maximal run of disks which ends at $D_{2k+1}$ and is contained in $D_{f_{k+1}}$, includes $D_{2k+1}$ only, since $D_{2k} \notin D_{f_{k+1}}$. In addition, $D_{2k+1}$ does not belong to $D_{f_k}$, so in particular $a_{2k+1} \notin S_{f_k}$. Hence, $a_{2k+1} \notin S_{f_{k+1}}$, because there is no transition (in this setup) from $(f_k', f_{k+1}')$ to $(f_{k+1}, S_{f_{k+1}})$ such that $a_{2k+1} \in S_{f_{k+1}}$. From this point on, the path is fully outside of $2D_k$, so, as easily verified by induction, $a_{2k+1}$ will not appear in any of the following states, including the state $(f_m, S_{f_m})$, as claimed. (The reader might wish to interpret this argument in terms of the actual moves of the frogs.)

Consider next an odd-indexed point $a_{2k+1} \in A$ 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 $a_{2k+1}$; this is because, from this point on, the B-frog remains inside $D_{2k+1}$ (note that, by construction, we do not execute the detour via $f'_k$), so the A-frog simply has to stay put at $a_{2k+1}$ and wait for the end of the sequence of moves of the B-frog.

Note that $f_0$ is contained in all blue disks and in $D_1$. In particular, this implies the asserted property for $k = 0$: The A-frog goes from the center of $D_0$ to the center of $D_1$ before the B-frog moves, and stays there till the end.

What the A-frog needs to do is to ensure that, for each $j \leq k$, it lies at the center of $D_{2j+2}$ 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 $j = 0$, because $f_0$ lies in $D_0, D_1, D_2$. For larger values of $j$, assume that the A-frog is at the center of $D_{2j}$ when the B-frog is at $f_{j-1}$. If the path goes straight to $f_j$, it exits $D_{2j-2}$ and then enters $D_{2j+1}$. Since the A-frog is at the center of $D_{2j}$, it can now move to the center of $D_{2j+1}$ and then to the center of $D_{2j+2}$, as desired. If the path goes to $f_j$ via $f'_{j-1}$, it exits $D_{2j-2}$ and $D_{2j-1}$, then re-enters $D_{2j-1}$ and then enters $D_{2j+1}$. However, since the A-frog is already at the center of $D_{2j}$, these additional exit and re-entry are irrelevant for it, and it can now move to the centers of $D_{2j+1}$ and $D_{2j+2}$ as above.

Copyright © SIAM.
Unauthorized reproduction of this article is prohibited.
Finally, when the $B$-frog moves to $f_k$, the $A$-frog, which is now at the center of $D_{2k}$, moves to the center of $D_{2k+1}$ and stays there. This completes the argument, and establishes the asserted exponential lower bound.

**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 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 [5, 6, 19].

### 8 Discussion

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, in subquadratic time, the continuous Fréchet distance between two polygonal curves, or the semi-continuous Fréchet distance between a polygonal curve and a sequence of points. The latter problem seems more amenable to the approach used in this paper and we are now exploiting this extension; the former problem seems much harder. (See [12] for recent results concerning the former problem.) It is also interesting to further reduce, if possible, the time bound of our algorithm, which is still rather close to quadratic (see the more detailed remark at the end of Section 7).

### References

[21] T. Feder and R. Motwani, Clique partitions, graph


