COMPUTING THE DISCRETE FRÉCHET DISTANCE IN SUBQUADRATIC TIME*

PANKAJ K. AGARWAL†, RINAT BEN AVRAHAM‡, HAIM KAPLAN§, AND MICHA SHARIR¶

Abstract. The *Fréchet distance* measures similarity between two curves f and g that takes into account the ordering of the points along the two curves: Informally, it is the minimum length of a leash required to connect a dog, walking along f, and its owner, walking along g, 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 m and n specific stones, respectively. The stones are in fact sequences of points, typically sampled from the respective curves f and g. 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 from the starting points to the terminal points of their sequences. The discrete Fréchet distance can be computed in $O(mn)$ 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 $O(\frac{mn \log\log n}{\log n})$ time, in the word 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.

Key words. Fréchet distance, similarity of curves, dynamic programming acceleration

AMS subject classifications. 68U05, 68W01, 65D18

1. Introduction. Problem statement. Let $P = (p_0, \ldots, p_{m-1})$ and $Q = (q_0, \ldots, q_{n-1})$ be two sequences of m and n points, respectively, in the plane. The *discrete Fréchet distance* between P and Q, denoted by $\mathrm{dF}(P, Q)$, is defined as follows. Fix a distance $\delta > 0$ and consider the directed graph $\mathcal{G}_\delta = \mathcal{G}_\delta(P, Q) = (P \times Q, E_P \cup E_Q \cup E_{PQ})$, where the edge sets E_P, E_Q, and E_{PQ} are defined as follows:

$$\begin{aligned} E_P =& \{((p_i, q_j), (p_{i+1}, q_j)) \mid 0 \leq i < m-1, 0 \leq j \leq n-1, \|p_i - q_j\|, \|p_{i+1} - q_j\| \leq \delta\}, \\ E_Q =& \{((p_i, q_j), (p_i, q_{j+1})) \mid 0 \leq i \leq m-1, 0 \leq j < n-1, \|p_i - q_j\|, \|p_i - q_{j+1}\| \leq \delta\}, \\ E_{PQ} =& \{((p_i, q_j), (p_{i+1}, q_{j+1})) \mid 0 \leq i < m-1, 0 \leq j < n-1, \|p_i - q_j\|, \|p_{i+1} - q_{j+1}\| \\ & \leq \delta\}; \end{aligned}$$

* Work by Pankaj Agarwal is supported by NSF under grants CCF-09-40671, CCF-10-12254, and CCF-11-61359, by ARO grants W911NF-07-1-0376 and W911NF-08-1-0452, and by an ERDC contract W9132V-11-C-0003. Work by Haim Kaplan is 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 is 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 is also supported by the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11). Work by Rinat Ben Avraham is supported by the Israel Science Fund Grants 338/09 and 822/10. A preliminary version of this paper has appeared in *Proc. 24th Annu. ACM-SIAM Sympos. Discrete Algorithms.*

†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

here we assume $\|\cdot\|$ to be the Euclidean norm. Then $\mathrm{dF}(P,Q)$ is the smallest $\delta > 0$ for which (p_{m-1}, q_{n-1}) is reachable from (p_0, q_0) in $\mathcal{G}_\delta$. Informally, think, as in the abstract, of P and Q as two sequences of stepping stones, and of two frogs, the P-frog and the Q-frog, where the P-frog has to visit all the P-stones in order, and the Q-frog has to visit all the Q-stones in order. The frogs are connected by a rope of length δ, and are initially placed at p_0 and q_0, 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 distances between the two frogs before and after the jump are both at most δ. The edges in E_P (resp., E_Q) correspond to the case when the P-frog (resp., Q-frog) jumps and the other stays at its current stone, and the edges in E_{PQ} correspond to the case when both frogs jump simultaneously. Then $\mathrm{dF}(P,Q)$ is the smallest $\delta > 0$ for which there exists a sequence of jumps that gets the frogs to p_{m-1} and q_{n-1}, respectively.

The continuous Fréchet distance. 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 $\alpha : [0,1] \to [0,1]$. In particular, $\alpha(0) = 0$ and $\alpha(1) = 1$. The Fréchet distance $\mathrm{F}(f,g)$ between two curves f and g is then defined as follows:

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

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

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 [29], speech recognition [31], signature verification [37], matching of time series in databases [30], map-matching of vehicle tracking data [9, 16, 40], and analysis of moving objects [10, 11]. 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 [34], handwriting recognition [39], and protein structure alignment [27, 41]. The Fréchet distance and its 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, and not just the proximity between the curves as unordered sets. For example, the two curves in Figure 1.1 have small Hausdorff distance because for every point on one curve, there is a nearby point on the other curve. However if the mapping between the two curves is required to be monotone, many points on one curve will have to be mapped to far away points on the other curve, and thus the Fréchet distance between them is large.

Eiter and Mannila [22] showed that the discrete Fréchet distance in the plane can be computed, using dynamic programming, in $O(mn)$ time. Aronov et al. [7] presented

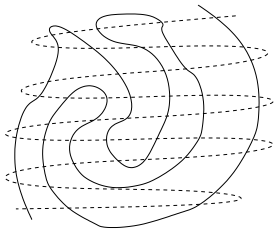

FIG. 1.1. *A pair of curves that are similar under the Hausdorff distance, but not similar under the Fréchet distance.*

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 [5] 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. [12] 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 [4] recently conjectured that the decision subproblem of the (continuous) Fréchet distance problem is 3SUM-hard [24]. Negative evidence for this conjecture has very recently been obtained by Buchin et al. [13]. They show that there exists an algebraic decision tree for the decision problem of depth $O(n^{2-\varepsilon})$, for some $\varepsilon > 0$. They have also extended our work to the continuous Fréchet distance and gave a randomized algorithm that runs in $O(n^2\sqrt{\log n}(\log\log n)^{3/2})$ expected time on a pointer machine and in $O(n^2(\log\log n)^2)$ expected time on a word RAM. Their algorithm requires elegant handling of the more involved details of the continuous case.

We note that no subquadratic algorithm in m and n, with any reasonable dependence on ε, is known even for computing a $(1+\varepsilon)$-approximation of either variant of the Fréchet distance between two arbitrary curves or two point sequences, with no restrictions on their shapes.

To date, the only subquadratic algorithms known for either the continuous or the discrete Fréchet distance problem are for restricted classes of curves, such as the algorithm of Aronov et al. [7] mentioned above. Other classes of curves considered so far in the literature include closed convex curves and *κ-bounded* curves [6]. 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. [6] 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 [4] 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. [21] 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 at most c times the radius of the ball.

Recently, Maheshwari et al. [33] gave an $O(nm)$-running time algorithm for the partial curve matching problem, improving upon a result of Alt and Godau [5]. In this problem, we are given two polygonal curves f and g, of m and n segments respectively, and a fixed parameter $\varepsilon \geq 0$, and we need to decide whether there exists a subcurve $f' \subseteq f$, such that $\mathrm{F}(f', g) \leq \varepsilon$. 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. [9, 40], by Buchin et al. [12] and, more recently, by Har-Peled and Raichel [25]. Other variants include the Fréchet distance between folded polygons, which are non-flat surfaces [17], and the Fréchet distance with shortcuts [20].

Our results. We present a new algorithm for computing the discrete Fréchet distance in the general unrestricted setting, whose running time is $O(mn \log\log n / \log n)$, assuming $m \leq n$, and which uses $O(n)$ storage. Our algorithm runs in the word RAM model. Thus, it is assumed that $O(\log n)$-bit integers can be stored using $O(1)$ space and can be accessed in $O(1)$ time. Some of the operations used in our algorithm exploit this model by encoding a small sequence of bits into a single word (regarding the bits as binary digits), and then use this word as an index to an array. We do not use any more involved bit manipulations, which are normally allowed in the word RAM model.

Our solution runs in two stages. We first present a procedure for solving the decision version of the problem: Given $\delta > 0$, determine whether $\mathrm{dF}(P, Q) \leq \delta$. The decision procedure runs in $O(mn \log\log n / \log^2 n)$ time and uses $O(n)$ space. Clearly, $\mathrm{dF}(P, Q)$ is one of the pairwise distances between the points of P and the points of Q. Thus, to obtain a solution for the optimization problem, we combine the decision procedure with a relatively simple binary search over the mn distances in $P \times Q$, without enumerating all of them explicitly, based on a simple, and significantly subquadratic procedure of Agarwal et al. for distance selection [1]. This increases the total running time by only a factor of $O(\log n)$, so the overall algorithm for computing $\mathrm{dF}(P, Q)$ 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, as promised, linear in $m + n$. The following presentation is therefore mainly focused on the decision procedure, which is the crux of our algorithm.

We note that similar improvements have been obtained for quadratic solutions of several other problems. For instance, Baran et al. [8] 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 running-time barrier are for computing the *edit distance* between two strings and for computing the most probable sequence of states in a hidden Markov model [18, 26, 32, 35]. Another similar improvement was obtained for the all-pairs shortest path problem in [14]. See also [23] for an earlier smaller improvement for this latter 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.

The specific way in which geometry is used in the decision procedure is by forming an arrangement of disks of radius δ centered at the points of a small subsequence, called a *block*, of P, and by locating each point of Q in this arrangement. This leads to an encoding of the problem by discrete states, each consisting of a face f of the arrangement where the Q-frog lies, coupled with a subset of the P-disks that contain f, where the P-frog can only lie in one of their centers.

The above representation already leads, with some care, to an improved, subquadratic running time of the decision procedure. However, this improvement turns out to gain insufficient speedup for the overall optimization task. This is because, as mentioned above, finding the discrete Fréchet distance itself requires an extra layer of optimization that is implemented by a binary search over critical candidate values, making the overall cost of the algorithm $O(\log n)$ times the cost of the decision procedure. Without any further improvement, the overall resulting running time is therefore still (slightly) super-quadratic. To beat the quadratic time barrier for the problem of finding the discrete Fréchet distance itself, we further reduce the cost of the decision procedure by an additional logarithmic factor.

The second improvement is achieved by exploiting the geometry of the problem in a subtler manner, realized by performing a 2-stage clustering of the P-sequence, first into larger *layers* and then into smaller *blocks*. This allows us to perform some of the geometric computations, such as point location of the Q-points in the disk arrangements, only at the layer level. As we show, this eliminates an additional logarithmic factor.

2. Overview of the Algorithm. This section provides a brief overview of the decision procedure: given P, Q, and a parameter $\delta > 0$, determine whether $\mathrm{dF}(P, Q) \leq \delta$. 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 P (resp., of Q). An entry $M_{i,j}$ of M is equal to 1 if there is a directed path from (p_0, q_0) to (p_i, q_j) in the graph $\mathcal{G}_\delta$, as defined in the beginning of the introduction, and to 0 otherwise. In other words, $M_{i,j}$ is 1 if the pair (p_i, q_j) is *reachable* from the starting placement (p_0, q_0) of the trip with a "leash" of length δ. The goal is to compute the value of $M_{m-1,n-1}$.

$M_{m-1,n-1}$ can be obtained by computing all entries of M using dynamic programming, as follows. If $\|p_0 - q_0\| \leq \delta$, we set $M_{0,0} \coloneqq 1$; otherwise, $M_{0,0} \coloneqq 0$ 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 $0 < j \leq n-1$ we set $M_{0,j} \coloneqq 1$ if $M_{0,j-1} = 1$ and $\|p_0 - q_j\| \leq \delta$; otherwise we set $M_{0,j} \coloneqq 0$. Clearly, if $M_{0,j} = 0$, for some $0 \leq j \leq n-1$, 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 $0 < i \leq m-1$ in order, $M_{i,0} \coloneqq 1$ if $M_{i-1,0} = 1$ and $\|p_i - q_0\| \leq \delta$; otherwise, we set $M_{i,0} \coloneqq 0$, and again all subsequent entries of the column will also be 0. For an arbitrary entry, $0 < i \leq m-1, 0 < j \leq n-1$, we set $M_{i,j} \coloneqq 1$ if at least one of $M_{i-1,j-1}$, $M_{i,j-1}$, and $M_{i-1,j}$ is 1, and $\|p_i - q_j\| \leq \delta$; otherwise, we set $M_{i,j} \coloneqq 0$. The cost of this dynamic programming procedure is $O(mn)$. Note that, as already mentioned, a simple modification of this procedure can compute, also in $O(mn)$ time, the actual discrete Fréchet distance itself [22].

To obtain a subquadratic decision procedure, we cannot compute each value of M explicitly, and instead we only compute 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 collection of rows of M into $t = \lceil m/\mu \rceil$ *blocks* $\mathsf{B}_0, \mathsf{B}_1, \ldots, \mathsf{B}_{t-1}$ each of length at most $\mu + 1$, such that the first row of the ith block is the last row of the previous block, for $i \geq 1$. More precisely, let R_j denote the jth row of M. Then B_i, the ith block of M, consists of the rows $R_{i\mu}, \ldots, R_{(i+1)\mu}$, for $0 \leq i < t-1$, and B_{t-1} consists of $R_{(t-1)\mu}, \ldots, R_{m-1}$. See Figure 2.1 (left). This corresponds to dividing P into t subchains $\Pi_0, \ldots, \Pi_{t-1}$, each of length at most $\mu + 1$, where $\Pi_i = (p_{i\mu}, \ldots, p_{(i+1)\mu})$, for $i < t-1$, and $\Pi_{t-1} = (p_{(t-1)\mu}, \ldots, p_{m-1})$. Let Φ_i and $\tilde{\Phi}_i (= \Phi_{i+1})$ denote the first and the last row of B_i, respectively.

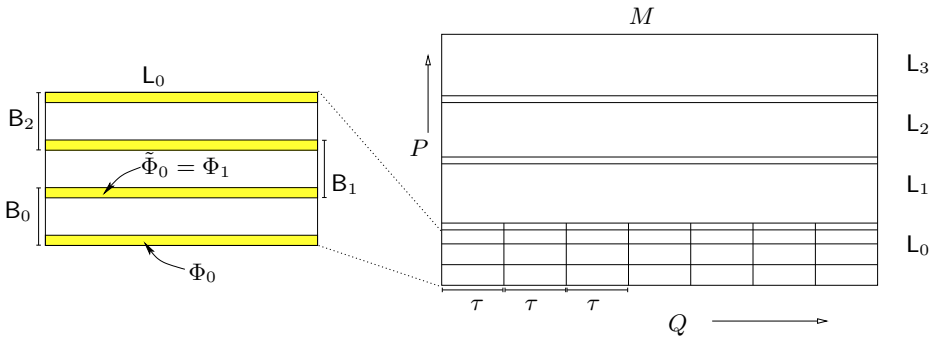

FIG. 2.1. *A partition of M into layers (right) and a partition of a layer into blocks (left). Q is partitioned into subchains of length τ. The automaton $\mathbb{K}_i^{\#}$ reads each subchain of Q along with the corresponding substring of Φ_i, each encoded as an integer, at each step.*

We first describe, in Section 3, an algorithm for processing each block B_i of M. We handle B_i using an approach that resembles the execution of a deterministic finite automaton $\mathbb{K}_i$. $\mathbb{K}_i$ has no accepting states. Instead, its goal is to produce the output string $\tilde{\Phi}_i$. In a preprocessing step for B_i we construct $\mathbb{K}_i$ in $O(\mu^5 2^\mu)$ time using $O(\mu^4 2^\mu)$ space. $\mathbb{K}_i$ is constructed (solely) from the corresponding subchain Π_i by exploiting its geometric structure, independently of Q. Next, we encode Q into a string over a finite, though not constant, alphabet; this step takes $O(n \log \mu) = O(n \log \log n)$ time. Informally, the encoding assigns to each point $q \in Q$ the face that contains q in the arrangement of the disks of radius δ centered at the points of Π_i. Then, assuming that Φ_i, the first row of B_i, already has been computed, $\mathbb{K}_i$ reads Φ_i and the encoding of Q and computes $\tilde{\Phi}_i$. At each step, $\mathbb{K}_i$ reads one bit of Φ_i and one symbol of the encoded Q and computes one output bit of $\tilde{\Phi}_i$ in $O(1)$ time, for a total of $O(n)$ time.

By choosing the constant c_1 in the expression for μ appropriately, constructing $\mathbb{K}_i$ and running it on Φ_i and on the encoding of Q takes $O(n)$ time. However, as will be detailed in Section 3, the encoding of Q takes $O(n \log \log n)$ time. Summing over all blocks, the overall time spent in processing the blocks, including the time spent on encoding Q for each block, and deciding whether $\mathrm{dF}(P, Q) \leq \delta$ is $O(\frac{mn \log \log n}{\log n})$. We note that we choose $\mu = O(\log n)$ because the upper bound on the size of $\mathbb{K}_i$ is exponential in μ. In fact, as shown in Section 7, there exist point sequences P and Q for which the size of $\mathbb{K}_i$ is indeed exponential.

The above procedure already gives a subquadratic algorithm for the decision problem, but this does not lead to an overall subquadratic optimization algorithm because of the extra logarithmic factor incurred by the binary search of the optimization procedure. To obtain a subquadratic optimization procedure, we improve the running

time of the decision problem by another $\log n$ factor. This is achieved in two stages. First, as will be described in Section 4, we transform $\mathbb{K}_i$ into another automaton $\mathbb{K}_i^{\#}$ that reads, in a single step, $\tau = \lceil \frac{c_2 \log n}{\log \log n} \rceil$ consecutive points of Q and τ consecutive bits of Φ_i, for a suitable constant c_2, and outputs a sequence of τ bits of $\tilde{\Phi}_i$. Each of these inputs and outputs of $\mathbb{K}_i^{\#}$ is encoded as an integer of length $O(\log n)$. See Figure 2.1 (right). In other words, $\mathbb{K}_i^{\#}$ simulates τ steps of $\mathbb{K}_i$ in a single step. As will be shown, the size and the construction time of $\mathbb{K}_i^{\#}$ are both $2^{O(\mu+\tau)}$. By choosing the constants c_1, c_2 appropriately, $\mathbb{K}_i^{\#}$ can be constructed in $O(n^{1/2})$ time. Ignoring the time spent on encoding Q and Φ_i (addressed shortly), $\mathbb{K}_i^{\#}$ computes $\tilde{\Phi}_i$ in $O(n/\log n)$ time. Excluding the time spent in computing the encodings of Q, the total time spent in processing all the blocks is $O(mn/\log^2 n)$.

Next, we show, in Section 5, that the encoding of Q can be batched over multiple blocks and a common encoding for all these blocks can be computed once, by increasing only slightly the size of the automata $\mathbb{K}_i^{\#}$. More precisely, we group $\lambda = \lceil \log n \rceil$ consecutive blocks into a single *layer*. The jth layer, denoted L_j, consists of the blocks $\mathsf{B}_{j\lambda}, \ldots, \mathsf{B}_{(j+1)\lambda-1}$, and its last row is also the first row of the next layer; see Figure 2.1 (right). By locating the points of Q in the arrangement of the disks centered at the points of the entire layer, we can compute, in $O(n \log\log n)$ time, a single, common encoding of Q for all the blocks in L_j. We modify the corresponding automata $\mathbb{K}_i^{\#}$ so that they all work with the new encoding of Q. Assuming that this encoding is available, the time spent in computing $\tilde{\Phi}_i$, the last row of block B_i, is still $O(n/\log n)$. Since there are $\Theta(\log n)$ blocks in a layer, the total time spent on one layer, including the time spent in computing the encoding of Q, is $O(n \log\log n)$. There are $\lceil m/\lambda\mu \rceil = O(m/\log^2 n)$ layers, so the overall time spent in computing $\tilde{\Phi}_{t-1}$, namely, the reachability bits for the last row R_{m-1} of M, is $O(mn \log\log n/\log^2 n)$.

Putting everything together, we obtain an algorithm that solves the decision problem in $O(mn \log\log n/\log^2 n)$ time. Finally, to solve the optimization problem, we combine the decision procedure with a relatively simple binary search on the pairwise distances in $P \times Q$, based on a significantly subquadratic procedure of Agarwal et al. for distance selection [1]. More specifically, the algorithm of Agarwal et al. [1] computes the l-th smallest pairwise distance in a set P of n points, for a given value $0 < l < \binom{n}{2}$, in $O(n^{3/2} \log^{5/2} n)$ time. We use a bichromatic variant of this algorithm (see Section 6). This search for the optimal δ increases the total running time by a logarithmic factor, so $\mathrm{dF}(P, Q)$ can be computed in $O(mn \log\log n/\log n)$ time.

3. Handling a Single Block. In this section, we describe the algorithm for processing a single block B_i of the matrix M. To simplify the notation, we denote B_i simply as B, denote the subchain $\Pi_i = (p_{i\mu}, \ldots, p_{(i+1)\mu})$ of P corresponding to B as Π, and rewrite its elements as $(\pi_0, \ldots, \pi_\mu)$. Similarly, we let $\Phi = (\varphi_0, \ldots, \varphi_{n-1})$ denote the first row of B and let $\tilde{\Phi} = (\tilde{\varphi}_0, \ldots, \tilde{\varphi}_{n-1})$ denote the last row of B. That is, Φ is the row of B corresponding to the starting point π_0 of Π and $\tilde{\Phi}$ is the row of B corresponding to the terminal point π_μ of Π. If B is the first block of M, then

$$\varphi_i = \begin{cases} 1 & \text{if } \|p_0 - q_j\| \le \delta \text{ for all } j \le i, \\ 0 & \text{otherwise.} \end{cases}$$

If B is not the first block, then Φ is the output sequence $\tilde{\Phi}_{i-1}$ obtained from processing the previous block B_{i-1}. In either case, given Φ and Π (and Q), the goal is to compute $\tilde{\Phi}$.

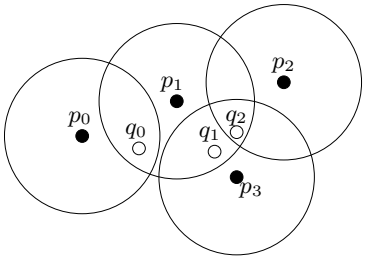

FIG. 3.1. *An illustration of the decision problem of the discrete Fréchet distance. The stepping stones of the P-frog are the black points. The disks of radius δ centered at the points of P form the arrangement $\mathcal{A}$. The stepping stones of the Q-frog are the hollow points. In this example, a legal path for the two frogs is* $\big((p_0,q_0),(p_1,q_0),(p_1,q_1),(p_1,q_2),(p_2,q_2),(p_3,q_2)\big)$.

We now show how to replace the sequence Q, which is a sequence of points in $\mathbb{R}^2$, by a sequence over a finite (albeit not constant) alphabet. For $0 \le j \le \mu$, let D_j be the disk of radius δ centered at π_j, and set $\mathcal{D} = \{D_j \mid 0 \le j \le \mu\}$. Construct the arrangement $\mathcal{A} = \mathcal{A}(\mathcal{D})$ of the disks in $O(\mu^2)$ time and preprocess it in $O(\mu^2 \log \mu)$ time, using any standard data structure (see [19]), such that point-location queries take $O(\log \mu)$ time. [1] Associate with each face f of $\mathcal{A}$ the subset $\mathcal{D}_f \subseteq \mathcal{D}$ of disks that contain f. Note that f can be of dimension 0, 1, or 2. Set $\Psi_f = \{\pi_j \mid D_j \in \mathcal{D}_f\}$. For any point $q \in f$, Ψ_f is equal to the subset of points of $\mathsf{\Pi}$ that are within distance δ from q. In other words, this set depends only on f and not on the choice of $q \in f$. Let $\mathcal{F}$ be the set of all faces in $\mathcal{A}$; we have $|\mathcal{F}| = O(\mu^2)$. For a point $q \in \mathbb{R}^2$, let $f(q)$ denote the face of $\mathcal{A}$ that contains q, and let $f_j = f(q_j)$, for $j = 0, \ldots, n-1$. A consequence of the observation made above is that the graph $\mathcal{G}_\delta(P,Q)$ does not depend on the exact coordinates of the points of Q but only on the faces of $\mathcal{A}$ that contain these points. More precisely, for any sequence $Q' = (q'_0, \ldots, q'_{n-1})$ of points such that $f(q'_j) = f_j$ for every $j = 0, \ldots, n-1$, $\mathcal{G}_\delta(P,Q')$ is the same as $\mathcal{G}_\delta(P,Q)$. Hence, we replace Q with the *face sequence* $F = (f_0, \ldots, f_{n-1})$. By performing a point-location query in $\mathcal{A}$ with each $q_j \in Q$, F can be computed in $O(n \log \mu) = O(n \log \log n)$ time.

Let $\mathsf{G}_\delta(P,F) = (P \times F, E)$ be the directed graph with edge set

$$E = \{((p_i,f_j),(p_k,f_l)) \mid ((p_i,q_j),(p_k,q_l)) \in (E_P \cup E_Q \cup E_{PQ})\}. \tag{3.1}$$

Let $\mathsf{G} = \mathsf{G}_\delta(\mathsf{\Pi},F)$ be the subgraph of $\mathsf{G}_\delta(P,F)$ induced by $\mathsf{\Pi} \times F$. Then, $\tilde{\varphi}_j = 1$ if there is a path in $\mathsf{G}_\delta(P,F)$ from (p_0,f_0) to (π_μ,f_j). Furthermore, the following statement holds.

$$\boxed{\begin{array}{l}\tilde{\varphi}_j = 1 \text{ iff } \exists k \le j \text{ such that } \varphi_k = 1 \text{ and} \\ \exists \text{ a path from } (\pi_0,f_k) \text{ to } (\pi_\mu,f_j) \text{ in } \mathsf{G}.\end{array}}$$

See Figure 3.2. In general, we call a pair $(\pi_i,f_j) \in \mathsf{\Pi} \times F$ *reachable* if there exists $k \le j$ such that $\varphi_k = 1$ and there is a path from (π_0,f_k) to (π_i,f_j) in G. We are thus interested in computing all reachable pairs in $\{\pi_\mu\} \times F$. Instead of computing the graph G explicitly and searching in G to find the reachable pairs of $\{\pi_\mu\} \times F$, we make the following observation. For any $0 \le j < n$, we define $\mathsf{X}_j \subseteq \mathsf{\Pi}$ as

$$\mathsf{X}_j = \{\pi \in \mathsf{\Pi} \mid (\pi,f_j) \text{ is reachable}\}. \tag{3.2}$$

[1] As mentioned in the overview in Section 2, performing point location in the disk arrangement of each block separately is not efficient enough. A more efficient procedure will be presented later in Section 5, but for now we use the separate arrangements, for the sake of simplicity of the presentation.

See Figure 3.3. Obviously, $\tilde{\varphi}_j = 1$ if and only if $\pi_\mu \in \mathsf{X}_j$. We show below that Π can be preprocessed into a finite automaton $\mathbb{K}$ such that, given X_j, f_j, f_{j+1}, and φ_{j+1}, $\mathbb{K}$ can quickly compute X_{j+1}. Then, $\mathbb{K}$ will process the faces of F in order, together with the corresponding bits of Φ, and will produce the output sequence $\tilde{\Phi}$.

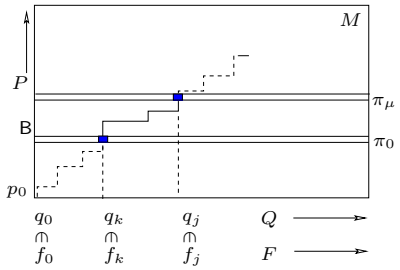

FIG. 3.2. *Passing through the block* B *from* (π_0, f_k) *to* (π_μ, f_j). *This path is meaningful only if its starting point* (π_0, f_k) *is reachable from* (p_0, f_0), *i.e.,* $\varphi_k = 1$.

We now describe the automaton $\mathbb{K} = (\Gamma, \Sigma, s, \Delta)$ of the current block, where Γ is the set of its states, Σ its alphabet, $s \in \Gamma$ its initial state, and Δ its transition function. Recall that $\mathbb{K}$ has no accepting states and its goal is to produce the output string $\tilde{\Phi}$. $\mathbb{K}$ computes $\tilde{\Phi}$ *on the fly*, one bit per transition. First, $\Sigma = \mathcal{F} \times \{0, 1\}$, i.e., at each step $\mathbb{K}$ reads one symbol from the face sequence F and one bit from Φ; clearly, $|\Sigma| = O(\mu^2)$. A state in Γ is represented as a pair $(f, \mathsf{S}_f) \in \mathcal{F} \times 2^{\Pi}$. To provide some intuition, a state (f, S_f) has the following interpretation: After reading a prefix of k faces of F along with the corresponding prefix of k bits of Φ, $\mathbb{K}$ enters the state (f, S_f) if and only if (i) $f = f_k$, and (ii) $\mathsf{S}_f = \mathsf{X}_k$. In other words, f is the kth face of F, and S_f contains exactly the points of Π that form a reachable pair with f_k. The initial state s is set to $(f, \emptyset)$ for some arbitrary face $f \in \mathcal{F}$. Note that $|\Gamma| = O(\mu^2 2^\mu)$.

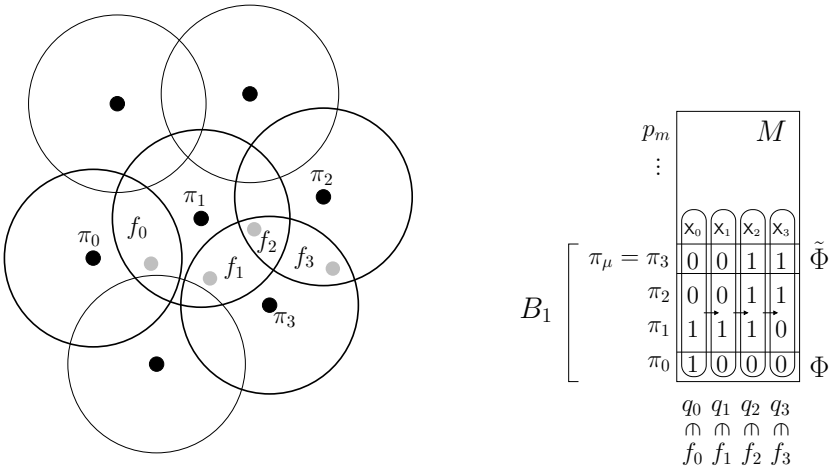

FIG. 3.3. *State transitions during processing of the first block. The disks of the first block, centered at* $\pi_0, \pi_1, \pi_2, \pi_3$, *are drawn thick. The states that* $\mathbb{K}$ *moves between when processing the first block are* $(f_0, \mathsf{X}_0), (f_1, \mathsf{X}_1), (f_2, \mathsf{X}_2), (f_3, \mathsf{X}_3)$ *in order. When encoded as bit vectors over the elements of* Π_0, $\mathsf{X}_0, \mathsf{X}_1, \mathsf{X}_2, \mathsf{X}_3$ *are the portions of the four respective columns of* M *within* B_0.

For a pair $((f, \mathsf{S}_f), (g, \varphi)) \in \Gamma \times \Sigma$, $\Delta((f, \mathsf{S}_f), (g, \varphi)) = ((g, \mathsf{S}_g), \tilde{\varphi})$ means that if

$\mathbb{K}$ is in state (f, S_f) and reads a face g and a bit φ it outputs the bit $\tilde{\varphi}$ and enters the state (g, S_g). An important thing to note is that, since we do not have the face sequence F in advance, we need to define the transition function Δ for all possible states (f, S_f) of $\mathbb{K}$ and for all possible pairs (g, φ) of a face and a reachability bit. We formally define the transition function $\Delta : \Gamma \times \Sigma \to \Gamma \times \{0, 1\}$, by specifying the *transition rules* that determine, for every state (f, S_f) and every input face g and input bit φ, the output state (g, S_g) and output bit $\tilde{\varphi}$ of Δ. In other words, this defines the transition rule $\Delta((f, \mathsf{S}_f), (g, \varphi)) = ((g, \mathsf{S}_g), \tilde{\varphi})$. Recall that $\Psi_f = \{\pi_j \mid D_j \in \mathcal{D}_f\}$.

(i) When the Q-frog jumps from f to g, the P-frog can either stay at a point $\pi_l \in \Psi_f$ if $\pi_l \in \Psi_g$ or jump from a point in Ψ_f to the next point of P if this point is in Ψ_g. Once the P-frog is at a point of Ψ_g, it can jump to the subsequent points that are in Ψ_g as long as it does not move to a point that is not in Ψ_g. Thus, the first transition rule is as follows. A point $\pi_l \in \Psi_g$ belongs to S_g if either $\pi_l \in \mathsf{S}_f$ or there exists $i < l$ such that $\pi_i \in \mathsf{S}_f$ and the entire run $\pi_{i+1}, \ldots, \pi_l$ is contained in Ψ_g.

(ii) Furthermore, if $\varphi = 1$ then (π_0, g) is a reachable pair of the previous block. Thus, the P-frog can be at π_0 while the Q-frog is at g. Then, as before, the P-frog can jump to the subsequent points that are in Ψ_g as long as it does not move to a point that is not in Ψ_g. Thus, if $\varphi = 1$, then S_g also contains the maximal prefix of points in Π that is contained in Ψ_g. That is, if ℓ is the smallest index such that $\pi_\ell \notin \Psi_g$ then $\pi_0, \ldots, \pi_{\ell-1}$ also belong to S_g.

(iii) If $\pi_\mu \in \mathsf{S}_g$ then (π_μ, g) is a reachable pair, and then $\tilde{\varphi} = 1$; otherwise $\tilde{\varphi} = 0$.

The transition from the state (f, S_f) to (g, S_g) after reaching a face $g \in F$ and a bit $\varphi \in \Phi$ occurs only if g is the successor of f in sequence F, and for every point $\pi_l \in \mathsf{S}_g$, and only for these points, at least one of the following conditions holds (in the graph G).

(i) The pair (π_l, f) is reachable and (π_l, g) can be reached from it by the edge $((\pi_l, f), (\pi_l, g))$ of G, so it is also reachable. Note that in this case $\pi_l \in \mathsf{S}_f$.

(ii) $\exists i < l$ such that the pair (π_i, f) is reachable and (π_l, g) can be reached from it by a path in G consisting of the edge $((\pi_i, f), (\pi_{i+1}, g))$, followed by the edges $((\pi_{i+1}, g), (\pi_{i+2}, g)), \ldots, ((\pi_{l-1}, g), (\pi_l, g))$, again making (π_l, g) also reachable. Note that here $\pi_i \in \mathsf{S}_f$.

(iii) If $\varphi = 1$, the pair (π_0, g) is reachable and (π_l, g) can be reached from it by the edges $((\pi_0, g), (\pi_1, g))$, $\ldots$, $((\pi_{l-1}, g), (\pi_l, g))$ of G.

The transition function Δ can be stored as a look-up table, called the *transition table*, of size $|\Gamma \times \Sigma| = O(\mu^4 2^\mu)$, which is indexed by an appropriate encoding of f, S_f, g, and the bit φ. Furthermore, each entry of Δ can be pre-computed in $O(\mu)$ time in a straightforward manner, according to the transition rules. That is, given f, S_f, g, and the bit φ, we can compute S_g using transition rules (i)-(ii) and we can compute $\tilde{\varphi}$ using transition rule (iii). Hence, $\mathbb{K}$ can be constructed in $O(\mu^5 2^\mu) = O(\log^5 n \cdot 2^{c_1 \log n})$ time. By choosing the constant c_1 to be at most $1/8$, we can ensure that $\mathbb{K}$ can be constructed in $O(n^{1/8} \log^5 n)$ time.

After having constructed $\mathbb{K}$, we run it on the face sequence F and the first row Φ of B, as follows. Initially, $\mathbb{K}$ is in the state s. Suppose that $\mathbb{K}$ 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}, φ_{j+1}). Suppose that

$$\Delta((f_j, \mathsf{S}_{f_j}), (f_{j+1}, \varphi_{j+1})) = ((f_{j+1}, \mathsf{S}_{f_{j+1}}), \tilde{\varphi}).$$

Then $\mathbb{K}$ sets $\tilde{\varphi}_{j+1} := \tilde{\varphi}$ and enters the state $(f_{j+1}, \mathsf{S}_{f_{j+1}})$. Note that the first move, from the initial state $s = (f, \emptyset)$, can be triggered only by a transition corresponding to

condition (iii), and not by crossing from f, which is some arbitrarily chosen face. Using an appropriate indexing of the states of $\mathbb{K}$ and using the transition-table representation of Δ, each step of $\mathbb{K}$ can be executed in $O(1)$ time. Hence, after computing the face sequence F and constructing $\mathbb{K}$, $\tilde{\Phi}$ can be produced in $O(n)$ time. Adding the time required for constructing the face sequence, the above processing of B can be performed in $O(n \log\log n)$ time using $O(n)$ space.

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

LEMMA 3.1. *Let P and Q be two sequences of points of length m and n, respectively, in $\mathbb{R}^2$, with $m \leq n$, and let $\delta > 0$ be a parameter. The decision problem, i.e., determining whether $\mathrm{dF}(P,Q) \leq \delta$, can be solved in $O\left(\frac{mn \log\log n}{\log n}\right)$ time using $O(n)$ space.*

Remark. We choose $\mu = O(\log n)$ because the size of $\mathbb{K}$ is exponential in μ. As shown in Section 7, there exists a point sequence P for which the size of $\mathbb{K}$ is indeed exponential, in the sense that there exist exponentially many point sequences Q that lead to these exponentially many states. This lower bound, however, does not preclude the possibility of constructing a more clever automaton, whose size is polynomial in μ, by exploiting further properties of the discrete Fréchet distance. The existence, and efficient construction, of such an automaton has so far remained elusive for us. See also a related remark at the end of Section 7.

4. Compacting Q. In this section we describe a different automaton for processing a block B that computes $\tilde{\Phi}$ faster, provided the face sequence F as well as the first row Φ is given in a "compressed" form. We will follow the notation from the previous section.

Let $\tau = \lceil \frac{c_2 \log n}{\log\log n} \rceil$ be a parameter, where c_2 is a constant to be chosen later. For simplicity, we assume that n is divisible by τ. The modified automaton, denoted $\mathbb{K}^{\#}$, reads in each step a string of τ consecutive faces of the face sequence F and the corresponding string of τ consecutive bits of Φ, and outputs the τ corresponding bits of $\tilde{\Phi}$. Namely, we partition F into $u = \lceil n/\tau \rceil$ substrings $F_1, \ldots, F_u$, of size τ each, and encode each F_i as a β-bit integer $e(F_i)$, for $\beta = O(\log n)$, as described below. We also partition Φ into u corresponding substrings $\boldsymbol{\varphi}_1, \ldots, \boldsymbol{\varphi}_u$, each of size τ, and view each $\boldsymbol{\varphi}_i$ as a τ-bit integer $e(\boldsymbol{\varphi}_i)$. Similarly, let $\tilde{\boldsymbol{\varphi}}_1, \ldots, \tilde{\boldsymbol{\varphi}}_u$ be the partition of $\tilde{\Phi}$ into u strings of length τ each, and treat each $\tilde{\boldsymbol{\varphi}}_i$ as a τ-bit integer $e(\tilde{\boldsymbol{\varphi}}_i)$. $\mathbb{K}^{\#}$ will compute $e(\tilde{\boldsymbol{\varphi}}_1), \ldots, e(\tilde{\boldsymbol{\varphi}}_u)$. If B is the first block, then $e(\boldsymbol{\varphi}_1) \ldots e(\boldsymbol{\varphi}_u)$ can be computed in $O(n)$ time; otherwise they will be the output of the automaton that processed the previous block. Since $e(F_i)$ and $e(\boldsymbol{\varphi}_i)$ 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 the word RAM model. In each step, the new automaton $\mathbb{K}^{\#}$ will read $e(F_i)$, $e(\boldsymbol{\varphi}_i)$ and output $e(\tilde{\boldsymbol{\varphi}}_i)$, in $O(1)$ time.

We now describe the structure of $\mathbb{K}^{\#} = (\Gamma, \Sigma^{\#}, s, \Delta^{\#})$, which like $\mathbb{K}$ is constructed independently of F and Φ. $\mathbb{K}^{\#}$ has the same set of states Γ as $\mathbb{K}$, and its initial state s is the same as that of $\mathbb{K}$. The alphabet $\Sigma^{\#}$ is $[0 : 2^{\beta-1}] \times [0 : 2^{\tau-1}]$, i.e., each symbol in $\Sigma^{\#}$ is a pair of integers of bit-length β and τ, respectively, intended to encode a pair $(F_i, \boldsymbol{\varphi}_i)$, where, as above, F_i is a sequence of τ faces and $\boldsymbol{\varphi}_i$ is a string of τ bits.

The transition function $\Delta^{\#}$ of $\mathbb{K}^{\#}$ maps $\Gamma \times \Sigma^{\#}$ to $\Gamma \times [0 : 2^{\tau-1}]$, and is defined as follows. Let (f, S_f) be a state of $\mathbb{K}^{\#}$, $\Xi = (\xi_1, \ldots, \xi_\tau)$ a sequence of τ faces of $\mathcal{F}$, and $\mathbf{b} = (\mathsf{b}_1, \ldots, \mathsf{b}_\tau)$ a sequence of τ bits. To define $\Delta^{\#}((f, \mathsf{S}_f), (e(\Xi), e(\mathbf{b})))$, we run $\mathbb{K}$ on Ξ and $\mathbf{b}$ starting at the state (f, S_f). If $\mathbb{K}$ outputs the string $\tilde{\mathbf{b}} = (\tilde{\mathsf{b}}_1, \ldots, \tilde{\mathsf{b}}_\tau)$

and enters the state $(\xi_\tau, \mathsf{S}_{\xi_\tau})$ after processing the sequence of τ matching pairs from Ξ and $\mathbf{b}$, then we set $\Delta^{\#}((f, \mathsf{S}_f), (e(\Xi), e(\mathbf{b}))) = ((\xi_\tau, \mathsf{S}_{\xi_\tau}), e(\tilde{\mathbf{b}}))$.

Let $\mathcal{F}^\tau$ denote the family of all sequences of τ faces in $\mathcal{F}$. To construct $\mathbb{K}^{\#}$, we first construct $\mathbb{K}$. Then, for each state $(f, \mathsf{S}_f) \in \mathbb{K}$, for each τ-length face sequence $\Xi \in \mathcal{F}^\tau$, and for each τ-bit binary string $\mathbf{b}$, we compute $\Delta^{\#}((f, \mathsf{S}_f), (e(\Xi), e(\mathbf{b})))$ by running $\mathbb{K}$ on Ξ and $\mathbf{b}$ starting from the state (f, S_f), as just described. We store $\Delta^{\#}$ as a look-up table whose number of entries is equal to the number of states of $\mathbb{K}^{\#}$ multiplied by the number of possible input symbols for $\mathbb{K}^{\#}$. That is, the size of the transition table of $\Delta^{\#}$ is $O(\mu^2 2^\mu) \times 2^{\beta+\tau} = O(\mu^2 2^{\mu+\beta+\tau})$. Since the number of faces of $\mathcal{A}$ is $O(\mu^2) = O(c_1^2 \log^2 n) \leq c \log^2 n$, for a suitable constant c, each face can be encoded using at most $\lceil \log(c \log^2 n) \rceil = \lceil 2 \log\log n + \log c \rceil$ bits, so a τ-long face sequence requires at most $\beta = \tau \lceil 2 \log\log n + \log c \rceil \leq c' \log n$ bits, for a suitable constant c' proportional to c_2. By choosing c_1 and c_2 sufficiently small, we can make the size of $\Delta^{\#}$ substantially sublinear, for example $\sqrt{n}$.

After having constructed $\mathbb{K}$, each entry of $\Delta^{\#}$ can be computed in $O(\tau)$ time from the transition table Δ in the following manner. Given a state (f, S_f), a sequence $\Xi = (\xi_1, \ldots, \xi_\tau)$ of τ input faces, and a sequence $\mathbf{b} = (\mathsf{b}_1, \ldots, \mathsf{b}_\tau)$ of τ input bits, first retrieve the entry of Δ indexed by $((f, \mathsf{S}_f), (\xi_1, \mathsf{b}_1))$ obtaining the state $(\xi_1, \mathsf{S}_{\xi_1})$ and output bit $\tilde{\mathsf{b}}_1$ of $\mathcal{K}$ by , then retrieving the entry of Δ indexed by $((\xi_1, \mathsf{S}_{\xi_1}), (\xi_2, \mathsf{b}_2)$ obtaining the state $(\xi_2, \mathsf{S}_{\xi_2})$ and output bit $\tilde{\mathsf{b}}_2$ of $\mathbb{K}$, and so on, until finally retrieving the entry of Δ indexed by $((\xi_{\tau-1}, \mathsf{S}_{\tau-1}), (\xi_\tau, \mathsf{b}_\tau)$ and obtaining the state $(\xi_\tau, \mathsf{S}_{\xi_\tau})$ and output but $\tilde{\mathsf{b}}_\tau$ of $\mathbb{K}$. We then set $\Delta^{\#}((f, \mathsf{S}_f), (e(\Xi), e(\mathbf{b})))$ to be $((\xi_\tau, \mathsf{S}_{\xi_\tau}), e(\tilde{\mathsf{b}}_1, \ldots, \tilde{\mathsf{b}}_\tau)$ using the encoding scheme described next. By choosing c_1 and c_2 sufficiently small, the total construction time of $\Delta^{\#}$ can also be made to be smaller than $n^{1/2}$, for example.

To complete the description of $\mathbb{K}^{\#}$, we now describe a simple encoding scheme that converts a τ-long string $\Xi = \langle \xi_1, \ldots, \xi_\tau \rangle \in \mathcal{F}^\tau$ to a β-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 $\nu = \lceil \log_2 |\mathcal{F}| \rceil = O(\log \mu)$ bits. We then put

$$e(\Xi) = \sum_{i=1}^{\tau} e(\xi_i) \cdot 2^{\nu(i-1)}. \tag{4.1}$$

That is, $e(\Xi)$ is the concatenation of the bit encodings of $e(\xi_1), \ldots, e(\xi_\tau)$. As already noted, the length of $e(\Xi)$ is

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

and $e(\Xi)$ can be computed in $O(\tau)$ time. Set

$$e(F) = (e(F_1), \ldots, e(F_u)) \quad \text{and} \quad e(\Phi) = (e(\boldsymbol{\varphi}_1), \ldots, e(\boldsymbol{\varphi}_u)).$$

Given $\mathbb{K}^{\#}, e(F)$, and $e(\Phi)$, we can compute $e(\tilde{\Phi})$, that is defined analogously to $e(\Phi)$, by running $\mathbb{K}^{\#}$ on $e(F)$ and $e(\Phi)$. Suppose $\mathbb{K}^{\#}$ is in state $\gamma_{i-1} \in \Gamma$ after reading the first $i-1$ integer pairs of matching entries of $e(F)$ and $e(\Phi)$. In the next step, $\mathbb{K}^{\#}$ reads the pair of integers $(e(F_i), e(\boldsymbol{\varphi}_i))$. If $\Delta^{\#}(\gamma_{i-1}, (e(F_i), e(\boldsymbol{\varphi}_i))) = (\gamma_i, \mathbf{b})$, then $\mathbb{K}^{\#}$ outputs the integer $\mathbf{b} = e(\tilde{\boldsymbol{\varphi}}_i)$ and enters the state γ_i. The ith 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 the already available indices γ_{i-1} and $(e(F_i), e(\boldsymbol{\varphi}_i))$. Hence, after

having constructed $\mathbb{K}^{\#}$ and the encodings $e(F)$ and $e(\Phi)$, $e(\tilde{\Phi})$ can be computed in $O(u) = O(\frac{n \log\log n}{\log n})$ time. Since the procedure just sketched obviously uses only $O(n)$ storage, and since $\mathbb{K}^{\#}$ can be constructed in $O(n^{1/2})$ time, we obtain the following summary result of this section.

LEMMA 4.1. *Let B be a block of M and Π the subchain corresponding to B. Given $e(F)$, the encoding of the face sequence corresponding to Q, and $e(\Phi)$, the encoding of the first row of B, the encoding $e(\tilde{\Phi})$ of the last row can be computed, in the word RAM model, in $O\left(\frac{n \log\log n}{\log n}\right)$ time using $O(n)$ space.*

5. Handling a Layer. Lemma 4.1 already provides the bound that we are after, but it relies on the availability of the encoding of Q that it needs to use. In this section we describe how to compute this encoding of Q more efficiently by batching it over multiple blocks. This, combined with the tools developed above, will finally lead to an overall $O(mn \log\log n/\log^2 n)$ algorithm for the decision problem.

Put $\lambda = \lceil \log n \rceil$. We partition the blocks $\mathsf{B}_0, \ldots, \mathsf{B}_{t-1}$ into $\ell = \lceil t/\lambda \rceil$ layers, $\mathsf{L}_0, \ldots, \mathsf{L}_{\ell-1}$, where L_i contains the blocks $\mathsf{B}_{i\lambda}, \ldots, \mathsf{B}_{(i+1)\lambda-1}$, for $i = 0, \ldots, \ell-2$, and $\mathsf{L}_{\ell-1}$ contains the remaining blocks $\mathsf{B}_{(\ell-1)\lambda}, \ldots, \mathsf{B}_{t-1}$. For simplicity, we only describe the procedure for the layer L_0, but the same procedure works for any other layer. We first describe how to compute a single encoding of Q for all blocks in L_0, and then describe the construction of the automaton for each block in L_0.

Encoding of Q. For simplicity, we use L to denote the layer L_0. L contains the blocks $\mathsf{B}_0, \ldots, \mathsf{B}_{\lambda-1}$. Let $\bar{\Pi} = (p_0, \ldots, p_{\lambda\mu})$ be the subchain of P spanned by all the blocks in L; we have $|\bar{\Pi}| = O(\mu\lambda) = O(\log^2 n)$. We construct the arrangement $\bar{\mathcal{A}}$ of the disks of radius δ centered at the points of $\bar{\Pi}$. That is, we construct one arrangement for the entire layer instead of a separate arrangement for each of its blocks. Let $\bar{\mathcal{F}}$ denote the set of faces in $\bar{\mathcal{A}}$; $|\bar{\mathcal{F}}| = O(\mu^2\lambda^2) \leq c \log^4 n$ where c is a constant that depends on c_1. We preprocess $\bar{\mathcal{A}}$, in $O(\log^4 n \log\log n)$ time, for answering point-location queries, using any of the standard techniques [19]; each query takes $O(\log\log n)$ time. Fix a block B_j, for some $0 \leq j < \lambda$. Let $\mathcal{A}_j$ be the arrangement constructed only on the disks centered at the points of its associated point sequence Π_j. Each face f of $\bar{\mathcal{A}}$ is a subface of some face $f^{(j)}$ of $\mathcal{A}_j$, for each j. See Figure 5.1. We find these correspondences by preprocessing each of the arrangements $\mathcal{A}_j$ for fast point location. For each face f of $\bar{\mathcal{A}}$, we pick an arbitrary point in f and locate it in each $\mathcal{A}_j$, thereby obtaining the coarser face $f^{(j)}$ of $\mathcal{A}_j$ containing f, and we store a pointer from $f^{(j)}$ to f. Thus, each "super-face" $f^{(j)}$ stores pointers to the faces of $\bar{\mathcal{A}}$ that lie inside $f^{(j)}$. For a face g of $\mathcal{A}_j$, let $\bar{F}_g$ denote the set of faces in $\bar{\mathcal{A}}$ that lie inside g.

Next, for each point q_i of the Q-sequence, we locate the face f_i of $\bar{\mathcal{A}}$ containing q_i, using the point-location structure. This takes $O(n \log\log n)$ time (now per layer and not per block). We obtain a sequence $\mathsf{F} = (\mathsf{f}_0, \mathsf{f}_1, \ldots, \mathsf{f}_{n-1})$ of faces of $\bar{\mathcal{A}}$, and we partition it into u subsequences $\mathsf{F}_1, \ldots, \mathsf{F}_u$, each consisting of τ consecutive faces, where $\tau = \lceil c_2 \log n/\log\log n \rceil$ and $u = \lceil n/\tau \rceil = \Theta(n \log\log n/\log n)$, as above.

Now comes the other improvement in the construction of the block-automata considered in Section 3. Specifically, we encode F in a manner similar to the previous section, except that we use the faces in $\bar{\mathcal{F}}$ instead of those in the coarser block subarrangements. Since $|\bar{\mathcal{F}}| \leq c \log^4 n$, each face f in $\bar{\mathcal{F}}$ can be labeled as an integer $\bar{e}(\mathsf{f}) \in [0 : c \log^4 n - 1]$ with

$$\bar{\nu} = \lceil \log |\bar{\mathcal{F}}| \rceil = \lceil \log(c \log^4 n)) \rceil \leq \lceil 4 \log\log n + \log c \rceil \tag{5.1}$$

bits. Note that $\bar{\nu}$ is roughly twice as large as ν, the length of the encoding of a face

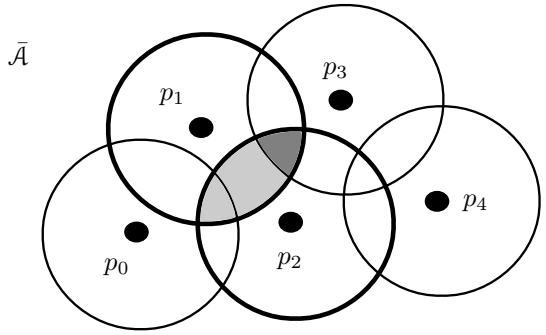

FIG. 5.1. *The arrangement $\bar{\mathcal{A}}$ of a layer is finer than each of the arrangements of its blocks. The disks drawn here all belong to the same layer. The disks of one of its blocks B_j are drawn thick. The entire shaded region is a face f of the arrangement of the disks of the block B_j. f consists of, and fully contains, several faces of $\bar{\mathcal{A}}$, one of which is depicted in darker gray.*

in the previous section. For each subsequence $\mathsf{F}_k = (\mathsf{f}_{(k-1)\tau}, \ldots, \mathsf{f}_{k\tau-1})$, $1 \le k \le u$, we compute its encoding

$$\bar{e}(\mathsf{F}_k) = \sum_{i=0}^{\tau-1} \bar{e}(\mathsf{f}_{(k-1)\tau+i}) \cdot 2^{\bar{\nu} i}. \tag{5.2}$$

Let $\bar{\beta}$ denote the length of the binary string $\bar{e}(\mathsf{F}_k)$. Note that $\bar{\beta} = \tau\bar{\nu} = O(\log n)$, and thus $\bar{e}(\mathsf{F}_k)$ can be stored in $O(1)$ space and accessed in $O(1)$ time, provided that we choose c_2, as above, sufficiently small. As earlier, $\bar{e}(\mathsf{F}_k)$ can be computed in $O(\tau)$ time. We set

$$\bar{e}(\mathsf{F}) = (\bar{e}(\mathsf{F}_1), \ldots, \bar{e}(\mathsf{F}_u)), \tag{5.3}$$

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

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

Processing a block in L. Suppose we have processed the blocks $\mathsf{B}_0, \ldots, \mathsf{B}_{i-1}$ of L and computed $e(\tilde{\Phi}_0), \ldots, e(\tilde{\Phi}_{i-1})$. To process B_i, we preprocess it into an automaton, independent of Q and $\Phi_i = \tilde{\Phi}_{i-1}$, and then use it to compute $e(\tilde{\Phi}_i)$. Let $\mathbb{K}_i, \mathbb{K}_i^{\#}$ denote the automata constructed for the block B_i in Sections 3 and 4, respectively. We first describe an "uncompacted" automaton $\mathbb{H}_i$, analogous to $\mathbb{K}_i$, that in each step reads one face of F and one bit of Φ_i and outputs a bit of $\tilde{\Phi}_i$. We then transform $\mathbb{H}_i$ into another compacted automaton $\mathbb{H}_i^{\#}$ that reads $\bar{e}(\mathsf{F})$ and $e(\Phi_i)$, the encoded compressed sequences of F and Φ_i respectively, in "chunks" as in Section 4. In a sense, we repeat the construction of $\mathbb{K}_i^{\#}$ from $\mathbb{K}_i$, but there are a few technical issues that require a different treatment, and various parameters need to be adapted to the new setup.

The automaton $\mathbb{H}_i = (\bar{\Gamma}_i, \bar{\Sigma}_i, \bar{s}_i, \bar{\Delta}_i)$ is very similar to $\mathbb{K}_i$ except that it is defined over the faces in $\bar{\mathcal{F}}$, instead of the faces of the corresponding individual arrangement $\mathcal{A}_i$. Namely, $\bar{\Sigma}_i = \bar{\mathcal{F}} \times \{0, 1\}$, and each state in $\bar{\Gamma}_i$ is a pair $(\mathsf{f}, \mathsf{S}_\mathsf{f}) \in \bar{\mathcal{F}} \times 2^{\Pi_i}$ with the

same interpretation as for $\mathbb{K}_i$. We have $|\bar{\Sigma}_i| = O(\mu^2\lambda^2)$ and $|\bar{\Gamma}_i| = O(\mu^2\lambda^2 2^\mu)$; that is, we use more faces but still consider only the points of Π_i as potential elements of the sets S_f. The transition function $\bar{\Delta}_i : \bar{\Gamma}_i \times \bar{\Sigma}_i \to \bar{\Gamma}_i \times \{0,1\}$ is also defined analogously. Since each face $\mathsf{f} \in \bar{\mathcal{F}}$ is a subface of a face $f^{(i)} \in \mathcal{A}_i$, $\mathbb{H}_i$ is a *refinement* of $\mathbb{K}_i = (\Gamma_i, \Sigma_i, s_i, \Delta_i)$ in the sense that $\mathbb{K}_i$ can be obtained from $\mathbb{H}_i$ by partitioning the states of $\mathbb{H}_i$ into equivalence classes, each class corresponding to the subfaces in $\bar{\mathcal{F}}$ of the same face of $\mathcal{A}_i$, and then by compressing the states in each equivalence class into a single state. More precisely, for a face $\mathsf{f} \in \bar{\mathcal{F}}$ lying in a face $f^{(i)}$ of $\mathcal{A}_i$, there is a state $(\mathsf{f}, \mathsf{S}_\mathsf{f}) \in \bar{\Gamma}_i$ if and only if $(f^{(i)}, \mathsf{S}_\mathsf{f}) \in \Gamma_i$. Similarly, for two faces $\mathsf{f}, \mathsf{g} \in \bar{\mathcal{F}}$ lying in the faces of $f^{(i)}$ and $g^{(i)}$ of $\mathcal{A}_i$ respectively, $\bar{\Delta}_i((\mathsf{f}, \mathsf{S}_\mathsf{f}), (\mathsf{g}, \varphi)) = ((\mathsf{g}, \mathsf{S}_\mathsf{g}), \tilde{\varphi})$ if and only if $\Delta_i((f^{(i)}, \mathsf{S}_\mathsf{f}), (g^{(i)}, \varphi)) = ((g^{(i)}, \mathsf{S}_\mathsf{g}), \tilde{\varphi})$.

We thus construct $\mathbb{H}_i$ from $\mathbb{K}_i$ as follows. For each state $(f, \mathsf{S}_f) \in \Gamma_i$, we follow the pointers from f to obtain a collection $\bar{F}_f$ of the faces of $\bar{\mathcal{A}}$ contained in f. For every $\mathsf{f} \in \bar{F}_f$, we add the pair $(\mathsf{f}, \mathsf{S}_f)$ to $\bar{\Gamma}_i$. Similarly, for each entry $\Delta_i((f, \mathsf{S}_f), (g, \varphi)) = ((g, \mathsf{S}_g), \tilde{\varphi})$ in the transition table of $\mathbb{K}_i$, we obtain, as above, the set $\bar{F}_g$ of the faces of $\bar{\mathcal{A}}$ contained in g, and then, for every $\mathsf{f} \in \bar{F}_f$ and every $\mathsf{g} \in \bar{F}_g$, we add the entry $\bar{\Delta}_i((\mathsf{f}, \mathsf{S}_f), (\mathsf{g}, \varphi)) = ((\mathsf{g}, \mathsf{S}_g), \tilde{\varphi})$, to the transition table of $\mathbb{H}_i$. The size of $\bar{\Delta}_i$ is $|\bar{\Gamma}_i| \times |\bar{\Sigma}_i| = O(\mu^4\lambda^4 2^\mu)$, and the time required to construct $\mathbb{H}_i$ from $\mathbb{K}_i$ is proportional to this bound. The construction time of $\mathbb{K}_i$ is $O(\mu^5\lambda^4 2^\mu) = O(c_1^5 \log^9 n \cdot 2^{c_1 \log n})$. By choosing c_1 sufficiently small, as in the previous section, we can ensure that the total time spent in constructing $\mathbb{H}_i$ is $O(n^{1/2})$.

Next, we construct the automaton $\mathbb{H}_i^\#$ from $\mathbb{H}_i$ in the same manner as in Section 4. That is, when $\mathbb{H}_i^\#$ is at state $(\mathsf{f}, \mathsf{S}_\mathsf{f})$ and reads a pair of integers $\bar{e}(\mathsf{F}_k)$ and $e(\boldsymbol{\varphi}_k)$, where F_k is a τ-long sequence of faces in $\bar{\mathcal{F}}$ and $\boldsymbol{\varphi}_k$ is a τ-bit binary string, it moves to the state $(\mathsf{f}_\tau, \mathsf{S}_{\mathsf{f}_\tau})$, where f_τ is the last face of F_k, and outputs the integer $e(\tilde{\boldsymbol{\varphi}}_k)$, where $\mathsf{S}_{\mathsf{f}_\tau}$ and $e(\tilde{\boldsymbol{\varphi}}_k)$ are both obtained by running the new $\mathbb{H}_i$ on F_k and $\boldsymbol{\varphi}_k$. The alphabet of $\mathbb{H}_i^\#$ consists of pairs of integers of respective lengths $\bar{\beta}$ and τ, and $\mathbb{H}_i^\#$ has $|\bar{\Gamma}_i|$ states, so the size of its transition table is

$$O(\mu^2\lambda^2 2^\mu) \times 2^{\bar{\beta}+\tau} = O(\mu^2\lambda^2 2^{\mu+\bar{\beta}+\tau}) = O(\log^4 n \cdot 2^{O(\log n)}), \tag{5.4}$$

where the constant of proportionality in the exponent is proportional to $c_1 + c_2$. Moreover, each entry of $\bar{\Delta}_i$ can be computed in $O(\tau)$ time. Once again, choosing c_1 and c_2 sufficiently small, we can make both the size of $\mathbb{H}_i^\#$ and its construction cost significantly sublinear, for example $O(n^{1/2})$.

Finally, we run $\mathbb{H}_i^\#$ on the encoded sequences $\bar{e}(\mathsf{F}) = (\bar{e}(\mathsf{F}_1), \ldots, \bar{e}(\mathsf{F}_u))$ and $e(\Phi_i) = (e(\boldsymbol{\varphi}_1), \ldots, e(\boldsymbol{\varphi}_u))$ to generate the encoding $e(\tilde{\Phi}_i) = (e(\tilde{\boldsymbol{\varphi}}_1), \ldots, e(\tilde{\boldsymbol{\varphi}}_u))$ of $\tilde{\Phi}_i$, the compacted representation of the last row of the block B_i. In the jth step, $\mathbb{H}_i^\#$ reads $\bar{e}(\mathsf{F}_j)$, $e(\boldsymbol{\varphi}_j)$ and outputs $e(\tilde{\boldsymbol{\varphi}}_j)$, in $O(1)$ time. The total time spent in this phase is $O(u) = O(n \log\log n / \log n)$. Similar to the analysis in the previous sections, the storage used by the algorithm is $O(n)$.

Putting the pieces together. After constructing the encoding of Q for a layer L_i, in $O(n \log\log n)$ time, all the 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 $\lceil m/\mu\lambda \rceil = \Theta(m/\log^2 n)$ layers, the total time spent in processing all of them, and in computing the last element $M_{m-1,n-1}$ of M, is $O(\frac{mn \log\log n}{\log^2 n})$. We thus obtain the following summary result concerning the decision procedure.

THEOREM 5.1. *Let P, Q 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,*

where we want to determine whether $\mathrm{dF}(P,Q) \leq \delta$*, can be solved in* $O(\frac{mn \log\log n}{\log^2 n})$ *time using* $O(n)$ *space, in the word RAM model.*

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 δ, in which an edge is added to the graph $\mathcal{G}_\delta$ as δ increases, are the pairwise distances between the points of P and the points of Q. 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 $P \times Q$ 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 $O(n^{3/2} \log^{5/2} n)$, where n is the total number of points. This algorithm easily adapts to the "bichromatic" scenario, in which we consider only distances between the pairs in $P \times Q$.

More specifically, we use a variant of the simpler (sequential) decision procedure of [1], in which we are given a parameter δ and wish to count the number of pairs in $P \times Q$ at distance at most δ. For this, we partition the set P 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 Q. In processing such a subset P_i, we construct the arrangement of the disks of radius δ centered at the points of P_i, and locate the points of Q in this arrangement, exactly as in [1]. Altogether, this can easily produce the number of pairs in $P \times Q$ 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, omitting the fairly routine details, we solve the optimization version of the distance selection algorithm using parametric searching [36], increasing the running time to $O(n^{3/2} \log^3 n)$.[2]

Since we call the distance selection procedure and the decision procedure for the discrete Fréchet distance $O(\log(mn))$ times during the search, we obtain the following main result of the paper.

THEOREM 6.1. *Let* P*,* Q *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* P *and* Q *can be computed in* $O(\frac{mn \log\log n}{\log n})$ *time using* $O(n)$ *space, in the word RAM model.*

7. A Lower Bound on the Size of the Automaton. An interesting question that arises in the design of the decision algorithm is how large can the automaton $\mathbb{K}$, described in Section 3, be. That is, how many states and transition rules can it have in the worst case? In this section we show a lower bound of $2^{|P|/2}$ to the number of states required by $\mathbb{K}$. More precisely, given a parameter $\delta > 0$, we construct a sequence $P = p_0, p_1, \ldots, p_{2m-1}$ of $2m$ points for which $\mathbb{K}$ must have 2^m states, in order to be prepared for any face sequence F.

For $0 \leq i < 2m$, let D_i denote the disk of radius δ centered at p_i. The points of P lie on the x-axis in the right-to-left order $p_0, p_2, \ldots, p_{2m-2}, p_1, p_3, \ldots, p_{2m-1}$. We label the odd-indexed points and disks as *red* and the even-indexed points and disks as *blue*. All red points are placed sufficiently close to each other so that all red disks have a large common intersection; see Figure 7.1. The blue points are placed so that, for each $k = 0, \ldots, m-1$, D_{2k} intersects D_{2k+1} in a small lune but is disjoint from D_{2k+3} (the second condition is vacuous for $k = m-1$). Let $\mathcal{A}$ be the arrangement

[2] Although there are more efficient algorithms for distance selection, which run in close to $O(n^{4/3})$ time [1, 28], this simple-minded solution suffices for our purpose, and it has the advantage that it only uses linear storage. To be more accurate, the algorithm of [1] runs in $O(n^{4/3} \log^{8/3} n)$ expected time, and the algorithm of [28] runs in $O(n^{4/3} \log^2 n)$ time.

of $D_0, \ldots, D_{2m-1}$. For a face $f \in \mathcal{A}$, let $\Psi_f = \{p_j \mid f \subseteq D_j\}$. Note that the caps $D_{2k} \cap D_{2k+1}$ are faces of $\mathcal{A}$. Finally, let $\mathbb{K}$ be the automaton constructed for P for the given δ, as described in Section 3.

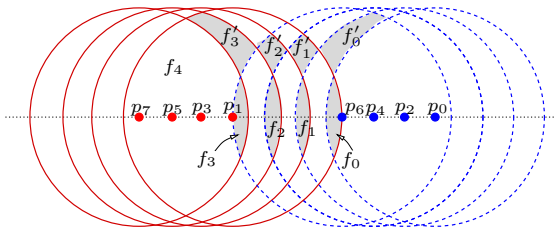

FIG. 7.1. *A sequence of points for which $\mathbb{K}$ has an exponential number of states. The red disks are drawn solid and the blue disks are drawn dashed. The faces $f_0, f'_0, \ldots, f_{m-1}, f'_{m-1}$ are highlighted.*

Let f_m be the face of $\mathcal{A}$ that is contained in all the red disks and is disjoint from all the blue disks. We claim that for each subset $\mathsf{S} \subset P$ of red points, $\mathbb{K}$ requires a state (f_m, S), which will imply the asserted exponential number of states. To be more precise, we claim that for any such S, we can construct a sequence F_S of $\Theta(m)$ faces with the following property: for any point $p_i \in \mathsf{S}$, as the Q-frog moves through the sequence F_S, the P-frog can execute a sequence of corresponding moves, so that it reaches p_i at the end, and this cannot be achieved, for the same F_S sequence, for any $p_i \notin \mathsf{S}$. In this argument we assume that P is the entire sequence, and not a subsequence of points in a single block. Thus we ignore the effect of the input string Φ that was used in the previous sections.

We now describe how to construct the desired F_S sequence for each subset S of red points. We begin by specifying the following $2m+1$ faces $f_0, f'_0, f_1, f'_1, \ldots, f_{m-1}, f'_{m-1}$, f_m of $\mathcal{A}$. For each $i = 0, \ldots, m-1$, we take f_i to be the cap $D_{2i} \cap D_{2i+1}$, which, by construction, is a face of $\mathcal{A}$; see Figure 7.1. 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 one needs to exit the disks D_{2i} and D_{2i+1} and does not have to cross the boundary of any other disk. Finally, we take f_m as defined above.

Given a subset S of red points, we construct a face sequence F_S, as follows. We start with the subsequence $(f_0, f_1, \ldots, f_{m-1}, f_m)$ and, for each red point $p_{2k+1} \notin \mathsf{S}$, we insert f'_k into F_S, between f_k and f_{k+1}. Figuratively, the sequence of jumps of the Q-frog, which proceeds from right to left, is a mixture of sharp vertical detours for red points not in S, and of short horizontal moves for red points in S. See Figure 7.2.

LEMMA 7.1. *After processing the sequence F_S, $\mathbb{K}$ reaches the state (f_m, S).*

Proof. Let (f_m, S_{f_m}) be the state that $\mathbb{K}$ reaches after reading the entire sequence Q. Consider first a red point $p_{2k+1} \notin \mathsf{S}$. When the Q-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 points of P which ends at p_{2k+1} and is contained in $\Psi_{f_{k+1}}$, includes p_{2k+1} only, since $p_{2k} \notin \Psi_{f_{k+1}}$. In addition, $p_{2k}, p_{2k+1} \notin \Psi_{f'_k}$, so in particular $p_{2k}, p_{2k+1} \notin \mathsf{S}_{f'_k}$. Hence, $p_{2k}, p_{2k+1} \notin \mathsf{S}_{f_{k+1}}$ because there is no transition (in this setup) from $(f'_k, \mathsf{S}_{f'_k})$ to $(f_{k+1}, \mathsf{S}_{f_{k+1}})$ such that $p_{2k+1} \in \mathsf{S}_{f_{k+1}}$. From this point on, the path is fully outside of D_{2k}, so, as easily verified by induction on the steps of $\mathbb{K}$, p_{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.

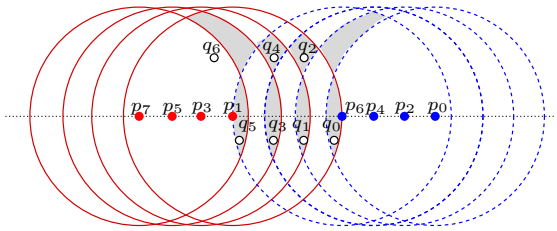

FIG. 7.2. $\mathsf{S} = \{1, 7\}$; *The points of* Q *are placed in* $f_0, f_1, f_1', f_2, f_2', f_3, f_4$ *in order, so that the* P*-frog can be at (a)* p_1 *and (b)* p_7 *when the* Q*-frog is done traversing the entire face sequence* F_{S}. *A sequence of moves of the two frogs that brings them to* (p_1, q_6) *is* $((p_0, q_0), (p_1, q_0), (p_1, q_2), (p_1, q_3), (p_1, q_4), (p_1, q_5), (p_1, q_6))$. *A sequence of moves of the two frogs that brings them to* (p_7, q_6) *is* $((p_0, q_0), (p_1, q_0), (p_2, q_1), (p_3, q_1), (p_4, q_1), (p_4, q_2), (p_4, q_3), (p_5, q_3), (p_6, q_3), (p_6, q_4), (p_7, q_5), (p_7, q_6))$. *No legal sequence of moves brings the two frogs to position* (p_3, q_6) *or to* (p_5, q_6).

Consider next a red point $p_{2k+1} \in \mathsf{S}$. For each $j \leq k$, we show that the configuration (p_{2j+1}, f_j), in which the P- and Q- frogs are at p_{2j+1} and f_j, respectively, is reachable from the initial configuration (p_0, f_0), in which the two frogs are at p_0 and f_0. This claim is obviously true for $j = 0$ because $f_0 \subset D_0, D_1$, so the Q-frog can remain at f_0 while the P-frog jumps to p_1 from p_0.

For larger values of j, assume that the configuration (p_{2j-1}, f_{j-1}) is reachable from (p_0, f_0). If $p_{2j-1} \in \mathsf{S}$, then $f'_{j-1} \notin F_{\mathsf{S}}$ and f_j appears right after f_{j-1} in F_{S}. When the Q-frog jumps from f_{j-1} to f_j it exits D_{2j-2}, enters D_{2j+1}, and remains inside D_{2j} during this jump. We can reach the configuration (p_{2j+1}, f_j) from (p_{2j-1}, f_{j-1}) in two steps: in the first step the P- and Q-frogs jump simultaneously to p_{2j} and f_j, respectively, and in the second step the Q-frog remains at f_j and the P-frog jumps to p_{2j+1}. Obviously both steps are valid moves.

Next, if $p_{2j-1} \notin \mathsf{S}$, then f'_{j-1} lies between f_{j-1} and f_j in F_{S}. As the Q-frog jumps from f_{j-1} to f'_{j-1} it exits D_{2j-2} and D_{2j-1}, and when it jumps to f_j it re-enters D_{2j-1} and enters D_{2j+1}. Note that both $f'_{j-1}, f_j \subset D_{2j}$. We can reach the configuration (p_{2j+1}, f_j) from (p_{2j-1}, f_{j-1}) in two steps: in the first step, the P- and Q-frogs jump sumltaneously to p_{2j} and f'_{j-1}, respectively, and in the second step they jump simulataneously to p_{2j+1} and f_j.

Hence, the configuration (p_{2k+1}, f_k) is reachable from (p_0, f_0) if and only if $p_{2k+1} \in \mathsf{S}$. Since $f'_k \notin F_{\mathsf{S}}$ and $f_{k+1}, f'_{k+1}, f_{k+2}, \ldots, f'_{m-1}, f_m \subset D_{2k+1}$, the P-frog can stay at p_{2k+1} and wait for the end of the sequence of moves of the Q-frog.

This completes the argument that a red point p_{2k+1} is in the final state, after reading F_{S}, if and only if $p_{2k+1} \in \mathsf{S}$, thereby implying that $\mathbb{K}$ reaches the state (f_m, S) after processing F_{S}. □

Changing the size of P back to m, we thus obtain the following result.

THEOREM 7.2. *For any* $\delta > 0$, *there exists a sequence* P *of* m *points in* $\mathbb{R}^2$ *so that the automaton constructed on* P *has more than* $2^{m/2}$ *states.*

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 F-sequences, or Q-sequences. Is it possible, for example, to reduce the number of states significantly by some sort of examination of the specific input Q-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 P, Q that guarantee that the number of states is much smaller. Earlier studies [6, 7, 21] gave efficient approximation algorithms for computing the Fréchet distance between special classes of curves and/or sequences. However, no exact algorithm that is subquadratic is known for the considered classes of curves. Showing that the number of states for specific sequences is small would hopefully give exact algorithms that are significantly subquadratic for computing the discrete Fréchet distance between such sequences.

8. Discussion. We have obtained an algorithm for computing the discrete Fréchet distance between two sets of points, which runs in subquadratic time and linear space, and which uses a deterministic finite automaton to encode and process legal positions of the frogs as they traverse the sequences P and Q.

Our algorithm can be extended to compute the discrete Fréchet distance between two sequences of points in $\mathbb{R}^d$, for any $d \geq 3$, in subquadratic time. We omit here the description of the fairly routine modifications that the algorithm requires, such as constructing arrangements of balls in $\mathbb{R}^d$ and performing point location in such arrangements [3, 15, 38]. While these operations are more expensive than their planar counterparts, they can still be implemented reasonably efficiently, because the number of balls in each arrangement is only $O(\log n)$.

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. As noted in the background, a similar extension to the continuous Fréchet distance, that has a slightly super-quadratic running time, was very recently obtained by Buchin et al. [13]. It is still interesting to know whether a general-purpose subquadratic running time can be achieved.

It is also interesting to further reduce, if possible, the running time of our algorithm, which is still rather close to quadratic. See the more detailed remark at the end of Section 7.

REFERENCES

[1] P. K. Agarwal, B. Aronov, M. Sharir, and S. Suri, *Selecting distances in the plane*, Algorithmica, 9(5) (1993), pp. 495–514.

[2] P. K. Agarwal, R. Ben Avraham, H. Kaplan, and M. Sharir, *Computing the discrete Fréchet distance in subquadratic time*, Proc. 24th Annu. ACM-SIAM Sympos. Discrete Algorithms, (2013), pp. 156–167.

[3] P. K. Agarwal and M. Sharir, *Arrangements and their applications*, in Handbook of Computational Geometry (J.-R. Sack and J. Urrutia, Eds.), Elsevier, (2000), pp. 49–119.

[4] 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), pp. 235–248.

[5] H. Alt and M. Godau, *Computing the Fréchet distance between two polygonal curves*, Internat. J. Comput. Geom. Appl., 5 (1995), pp. 75–91.

[6] H. Alt, C. Knauer, and C. Wenk, *Comparison of distance measures for planar curves*, Algorithmica, 38(1) (2004), pp. 45–58.

[7] B. Aronov, S. Har-Peled, C. Knauer, Y. Wang, and C. Wenk, *Fréchet distance for curves, revisited*, Proc. 14th Annu. European Sympos. Algorithms, (2006), pp. 52–63.

[8] I. Baran, E. D. Demaine, and M. Patrascu, *Subquadratic algorithms for 3SUM*, Algorithmica, 50(4) (2008), pp. 584–596.

[9] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk, *On map-matching vehicle tracking data*, Proc. 31st Intl. Conf. Very Large Data Bases, (2005), pp. 853–864.

[10] K. Buchin, M. Buchin, and J. Gudmundsson, *Detecting single file movement*, Proc. 16th ACM SIGSPATIAL Intl. Conf. Adv. GIS, (2008), pp. 288–297.

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

[12] 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), pp. 170–173.

[13] K. Buchin, M. Buchin, W. Meulemans, and W. Mulzer, *Four soviets walk the dog — with an application to Alt's conjecture*, available at: http://arxiv.org/pdf/1209.4403v1.pdf (2012).

[14] T. M. Chan, *More algorithms for all-pairs shortest paths in weighted graphs*, SIAM J. Comput., 39(5) (2010), pp. 2075–2089.

[15] B. Chazelle, *Cutting hyperplanes for divide-and-conquer*, Discrete Comput. Geom., 9 (1993), pp. 145–158.

[16] 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), pp. 75–83.

[17] A. F. Cook, A. Driemel, S. Har-Peled, J. Sherette, and C. Wenk, *Computing the Fréchet distance between folded polygons*, Proc. 12th Intl. Sympos. Algorithms and Data Structures, (2011), pp. 267–278.

[18] M. Crochemore, G. M. Landau, and M. Ziv-Ukelson, *A subquadratic sequence alignment algorithm for unrestricted scoring matrices*, SIAM J. Comput., 32 (2003), pp. 1654–1673.

[19] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd edition, Springer-Verlag, Berlin, 2008.

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

[21] A. Driemel, S. Har-Peled, and C. Wenk, *Approximating the Fréchet distance for realistic curves in near linear time*, Discrete Comput. Geom., 48(1) (2012), pp. 94–127.

[22] 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.

[23] T. Feder and R. Motwani, *Clique partitions, graph compression and speeding-up algorithms*, J. Comput. Sys. Sci., 51(2) (1995) pp. 261–272.

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

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

[26] D. Hermelin, G. M. Landau, S. Landau, and O. Weimann, *Unified Compression-Based Acceleration of Edit-Distance Computation*, Algorithmica, 65(2) (2013), pp. 339–353.

[27] M. Jiang, Y. Xu, and B. Zhu, *Protein structure-structure alignment with discrete Fréchet distance*, J. Bioinform. Comput. Biol., 6(1) (2008), pp. 51–64.

[28] M. J. Katz and M. Sharir, *An expander-based approach to geometric optimization*, SIAM J. Comput., 26(5) (1997), pp. 1384–1408.

[29] E. J. Keogh and M. J. Pazzani, *Scaling up dynamic time warping to massive datasets*, Proc. 3rd Euro. Conf. Princip. Data Mining and Know. Disc., (1999), pp. 1–11.

[30] M. S. Kim, S. W. Kim, and M. Shin, *Optimization of subsequence matching under time warping in time-series databases*, Proc. ACM Sympos. Applied Comput., (2005), pp. 581–586.

[31] S. Kwong, Q. H. He, K. F. Man, K. S. Tang, and C. W. Chau, *Parallel genetic-based hybrid pattern matching algorithm for isolated word recognition*, Intl. J. Pattern Recog. Art. Intel., 12(5) (1998), pp. 573–594.

[32] Y. Lifshits, S. Mozes, O. Weimann, and M. Ziv-Ukelson, *Speeding up HMM decoding and training by exploiting sequence repetitions*, Algorithmica, 54 (2009), pp. 379–399.

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

[34] A. Mascret, T. Devogele, I. Le Berre, and A. Hénaff, *Coastline matching process based on the discrete Fréchet distance*, Proc. 12th Intl. Sympos. Spatial Data Handling, (2006), pp. 383–400.

[35] W. J. Masek and M. S. Paterson, *A faster algorithm computing string edit distances*, J. Comput. Sys. Sci., 20(1) (1980), pp. 18–31.

[36] N. Megiddo, *Applying parallel computation algorithms in the design of serial algorithms*, J. ACM, 30 (1983), pp. 852–865.

[37] M. E. Munich and P. Perona, *Continuous dynamic time warping for translation-invariant curve alignment with applications to signature verification*, Proc. 7th Intl. Conf. Comp. Vision, (1999), pp. 108–115.

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

[39] E. Sriraghavendra, K. Karthik, and C. Bhattacharyya, *Fréchet distance based approach for searching online handwritten documents*, Proc. 9th Intl. Conf. Doc. Analy. Recog., (2007), pp. 461–465.

[40] C. Wenk, R. Salas, and D. Pfoser, *Addressing the need for map-matching speed: Localizing global curve-matching algorithms*, Proc. 18th Intl. Conf. Scientific and Statistical Database Management, (2006), pp. 379–388.

[41] B. Zhu, *Protein local structure alignment under the discrete Fréchet distance*, J. Comput. Biology, 14(10) (2007), pp. 1343–1351.