Fast Algorithms for Approximate Fréchet Matching Queries in Geometric Trees

Joachim Gudmundsson* Michiel Smid†

October 31, 2014

Abstract

Let T be a tree in $\mathbb{R}^d$ and let $\Delta > 0$ be a real number. The aim is to preprocess T into a data structure, such that for any polygonal query path Q, we can decide if T contains a path P whose Fréchet distance $\delta_F(Q, P)$ to Q is at most Δ. For any real number $\varepsilon > 0$, we present an efficient data structure that solves an approximate version of this problem for the case when T is c-packed and each of the edges of T and Q has length $\Omega(\Delta)$: If the query algorithm returns NO, then there is no such path P. If the query algorithm returns YES, then T contains a path P for which $\delta_F(Q, P) \leq (1+\varepsilon)\Delta$ if Q is a line segment, and $\delta_F(Q, P) \leq 3(1+\varepsilon)\Delta$ otherwise.

1 Introduction

The Fréchet distance [19] is a measure of similarity between two curves P and Q that takes into account the location and ordering of the points along the curves. Let p and p' be the endpoints of P and let q and q' be the endpoints of Q. Imagine a dog walking along P from p to p' and, simultaneously, a person walking along Q from q to q'. The person is holding a leash that is attached to the dog. Neither the dog nor the person is allowed to walk backwards along their curve, but they can change their speeds. The Fréchet distance between P and Q is the length of the shortest leash such that the dog can walk from p to p' and the person can walk from q to q'. To define this formally, assume the two curves are given as functions $P : [0, 1] \to \mathbb{R}^d$ and $Q : [0, 1] \to \mathbb{R}^d$, where $P(0) = p$, $P(1) = p'$, $Q(0) = q$, and $Q(1) = q'$. A *reparameterization* is an injective and continuous function $f : [0, 1] \to [0, 1]$; it is called *orientation-preserving* if $f(0) = 0$ and $f(1) = 1$. Let $|xy|$ denote the Euclidean

*School of IT, University of Sydney and NICTA, Australia. Research supported by the Australian Research Council (grant FT100100755). NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

†School of Computer Science, Carleton University, Ottawa, Canada. Research supported by NSERC.

distance between two points x and y. The Fréchet distance $\delta_F(P, Q)$ between the curves P and Q is defined as

$$\delta_F(P, Q) = \inf_f \max_{0 \le t \le 1} |P(f(t))Q(t)|,$$

where f ranges over all orientation-preserving reparameterizations.

Measuring the similarity between curves has been extensively studied in the last 20 years in computational geometry [1, 4, 18, 25], as well as in other areas, such as data mining [21, 23], GIScience [7, 10, 26] and image processing [24].

Alt and Godau [4] showed that the Fréchet distance between two polygonal paths P and Q can be computed in $O(n^2 \log n)$ time, where n is the total number of vertices of P and Q. A lower bound of $\Omega(n \log n)$ was given by Buchin *et al.* [11]. Despite extensive research, no subquadratic algorithm is known for the problem of computing the Fréchet distance. In 2009, Alt [2] conjectured the decision problem, i.e., for a given $\Delta > 0$, deciding if $\delta_F(P, Q) \le \Delta$, to be 3SUM-hard. Recently, Bringmann [8] showed that, assuming the Strong Exponential Time Hypothesis, the Fréchet distance cannot be computed in strongly subquadratic time, i.e., in time $O(n^{2-\delta})$ for any $\delta > 0$.

Agarwal *et al.* [1] showed how to achieve a subquadratic running time for the discrete Fréchet distance, where only the vertices of the paths are considered. Buchin *et al.* [12] showed how to extend their approach to the (continuous) Fréchet distance. Their decision algorithm takes $O(n^2(\log \log n)^{3/2}/\sqrt{\log n})$ expected time, which is subquadratic. Using this, they compute the exact Fréchet distance in $O(n^2\sqrt{\log n}(\log \log n)^{3/2})$ expected time.

Until recently, subquadratic algorithms for computing the Fréchet distance were only known for restricted cases, such as closed convex curves and κ-bounded curves [5]. Driemel *et al.* [18] introduced a new class of realistic curves, the so-called c-packed curves. A polygonal path is *c-packed* if the total length of the edges inside any ball is bounded by c times the radius of the ball. Note that this definition generalizes naturally to geometric graphs. Driemel *et al.* showed that a $(1+\varepsilon)$-approximation of the Fréchet distance between two c-packed curves with a total of n vertices in $\mathbb{R}^d$ can be computed in $O(cn/\varepsilon + cn \log n)$ time. Bringmann and Künnemann [9] improved the running time to $\widetilde{O}(cn/\sqrt{\varepsilon})$, which is optimal in high dimensions apart from lower order terms, unless the Strong Exponential Time Hypothesis fails. The notion of c-packedness has been argued [13, 18] to capture many realistic settings for geometric paths and graphs. For example, Chen *et al.* [13] experimentally verified that maps of real world cities are ϕ-low density for a constant ϕ[1]. A geometric graph G is *ϕ-low-density*, if for any radius $\rho > 0$, any ball with radius ρ intersects at most ϕ edges of G that are longer than ρ. A c-packed curve is ϕ-low density for $\phi = 2c$; see [18].

In many applications, it is important to find the path in a geometric graph G that is most similar to a polygonal curve Q. Alt *et al.* [3] introduced this problem in 2003: Given a planar geometric graph G with n vertices and a polygonal curve Q with m vertices, the problem is to find the path P on G that has the smallest Fréchet distance to Q. They presented an algorithm that finds such a path P, with both endpoints being vertices of G, in $O(nm \log(nm) \log n)$ time using $O(nm)$ space; see also [7, 26]. The bound on the running

[1] In their experiments, ϕ varied between 16 and 28.

time is close to quadratic in the worst case and, hence, unsuitable for large road maps. Chen *et al.* [13] considered the case when the embedding of G is ϕ-low density and the curve Q is c-packed. In $\mathbb{R}^d$, they presented a $(1+\varepsilon)$-approximation algorithm for the problem with running time $O((\phi m + cn)\log(nm)\log(n+m) + (\phi m/\varepsilon^d + cn/\varepsilon)\log(nm))$.

Very little is known about query variants of these problems. For any given polygonal path P with n vertices, Driemel and Har-Peled [17] gave a data structure of $O(n)$ size such that for any given query segment Q and any two query points x and y on P, a $(1+\varepsilon)$-approximation to the Fréchet distance between Q and the subpath of P between x and y can be computed in $O(\log n \log\log n)$ time. In the most general query setting, the aim is to preprocess a given geometric graph G into a data structure, such that for a polygonal path Q and a real value $\Delta > 0$ as a query, it can be decided if there exists a path P in G whose Fréchet distance to Q is at most Δ. In this setting, the path P does not have to start or end at a vertex of G. In [15], de Berg *et al.* studied the case when G is a polygonal path in $\mathbb{R}^2$ with n vertices and Q is a straight-line segment. For any fixed value of Δ and any parameter s with $n \leq s \leq n^2$, they show how to build, in $O(n^2 + s\,\mathrm{polylog}(n))$ time, a data structure of size $O(s\,\mathrm{polylog}(n))$, that can be used to approximately decide if G contains a path with Fréchet distance less than Δ to any given query segment. More precisely, for any query segment Q of length more than 6Δ, the query algorithm associated with the data structure returns YES or NO in $O((n/\sqrt{s})\,\mathrm{polylog}(n))$ time.

1. If the output is YES, then there exists a path P in G such that $\delta_F(Q,P) \leq (2+3\sqrt{2})\Delta$. (In fact, their algorithm counts all such "minimal" paths.)

2. If the output is NO, then $\delta_F(Q,P) > \Delta$ for any path P in G.

By increasing the preprocessing time to $O(n^3 \log n)$, they show that the same result holds for the case when the threshold Δ is part of the query.

Our results: We consider the same problem as de Berg *et al.* [15] for the case when the graph G is a tree and a query consists of a polygonal path Q. We will write T instead of G.

Let T be a tree with n vertices in $\mathbb{R}^d$. Thus, any vertex of T is a point in $\mathbb{R}^d$ and any edge is the line segment joining its two vertices. If $d = 2$, then this tree is not necessarily plane, i.e., edges of T may cross. A point x in $\mathbb{R}^d$ is said to be *on* T, if either x is a vertex of T or x is in the relative interior of some edge of T. If x and y are two points on T, then $T[x,y]$ denotes the path on T from x to y.

Let Δ be a fixed positive real number. We want to preprocess T such that queries of the following type can be answered efficiently: Given a polygonal path Q in $\mathbb{R}^d$ with m vertices, decide if there exist two points x and y on T, such that $\delta_F(Q, T[x,y]) \leq \Delta$.

Assume that the tree T is c-packed, for some constant c. Also, assume that each edge of T has length $\Omega(\Delta)$. For any constant $\varepsilon > 0$, we show that a data structure of size $O(n\,\mathrm{polylog}(n))$ can be built in $O(n\,\mathrm{polylog}(n))$ time. For any polygonal path Q with m vertices, each of whose edges has length $\Omega(\Delta)$, the query algorithm associated with the data structure returns YES or NO in $O(m\,\mathrm{polylog}(n))$ time.

1. If the output is YES and $m = 2$ (i.e., Q is a segment), then the algorithm also reports two points x and y on T such that $\delta_F(Q, T[x, y]) \leq (1 + \varepsilon)\Delta$.

2. If the output is YES and $m > 2$, then the algorithm also reports two points x and y on T such that $\delta_F(Q, T[x, y]) \leq 3(1 + \varepsilon)\Delta$.

3. If the output is NO, then $\delta_F(Q, T[x, y]) > \Delta$ for any two points x and y on T.

If T is a path, then we do not need the requirement that each of its edges has length $\Omega(\Delta)$.

Compared to the structure in [15], which only considers paths, the main drawbacks are that we require the input tree T to be c-packed and all its edges to have length $\Omega(\Delta)$. However, the advantages are that (1) our structure works in $\mathbb{R}^d$ for any constant $d \geq 2$, and can report a path $T[x, y]$, (2) the approximation bound is improved from $(2 + 3\sqrt{2})$ to $(1 + \varepsilon)$ for query segments, (3) our query algorithm can handle polygonal query paths, (4) the preprocessing time is improved from nearly quadratic to $O(n \operatorname{polylog}(n))$, and (5) the input graph can be a tree and is not restricted to being a polygonal path.

Organization: In Section 2, we consider a restricted version of the problem, in which a query consists of a line segment Q and two points x and y on the tree T, and we want to (approximately) decide if $\delta_F(Q, T[x, y]) \leq \Delta$. For the case when T is a polygonal path, Driemel and Har-Peled [17] have shown that, using $O(n)$ space and $O(n \log^2 n)$ preprocessing time, a $(1 + \varepsilon)$-approximation to $\delta_F(Q, T[x, y])$ can be computed in $O(\log n \log \log n)$ time, where the constant factors depend on ε. Since in our restricted problem, the value of Δ is fixed, we show that their approach can be modified such that the query time, for a polygonal path, becomes $O(\log n)$. We also show how to generalize the query algorithm to trees, resulting in a query time of $O(\log^2 n)$.

In Section 3, we present the general approach for querying a tree with a line segment, by giving a generic query algorithm and proving its correctness. Since the running time of this algorithm can be $\Omega(n^2)$ for arbitrary trees, we recall c-packed trees and μ-simplifications in Section 4 and prove some of their properties. In Section 5, we use μ-simplifications to show how the generic algorithm can be implemented efficiently for querying a polygonal c-packed path with a query segment. In Section 6, we generalize the result of Section 5 to c-packed trees, all of whose edges have length $\Omega(\Delta)$. In Section 7, we use the previous results to query polygonal paths and trees with a polygonal path. Finally, in Section 8, we conclude with some directions for future work.

Remark: In the preliminary version [20] of this paper, we presented data structures with approximation ratio $\sqrt{2}(1 + \varepsilon)$ for querying polygonal paths or trees with a query segment. These results hold in $\mathbb{R}^2$ and the space requirements and query times are a logarithmic factor worse than the ones presented in the current paper. However, their preprocessing times are faster by a logarithmic factor. The advantage of the results in [20] is that the presentation is completely self-contained. The current version heavily uses the results from Driemel and Har-Peled [17] that were mentioned before.

2 Approximate Subpath Fréchet Distance Queries

Let T be a tree with n vertices in $\mathbb{R}^d$, and let $\Delta > 0$ and $\varepsilon > 0$ be fixed real numbers. We consider queries of the following type: Given a line segment $Q = [a, b]$ and two points x and y on T (together with the edges of T that contain x and y), decide, in an approximate sense, if $\delta_F(Q, T[x, y]) \leq \Delta$. To be more precise, the query algorithm returns a Boolean value B that satisfies the following two properties:

1. If $B = \mathit{true}$, then $\delta_F(Q, T[x, y]) \leq (1 + \varepsilon)\Delta$.
2. If $B = \mathit{false}$, then $\delta_F(Q, T[x, y]) > \Delta$.

If T is a polygonal path, then we can use the data structure of Driemel and Har-Peled [17] to answer this type of query. This data structure has size $O(n)$, can be built in $O(n \log^2 n)$ time, and has a query time of $O(\log n \log \log n)$, where the constant factors depend on ε. In fact, this data structure can be used to compute a $(1 + \varepsilon)$-approximation to $\delta_F(Q, T[x, y])$. Since in our problem, the value of Δ is fixed, we show that their approach can be modified such that the query time becomes $O(\log n)$.

In Section 2.1, we consider the case when T is a polygonal path. In Section 2.2, we extend the solution to the case of trees.

2.1 Queries in Polygonal Paths

Throughout this section, we assume that T is a polygonal path in $\mathbb{R}^d$. We will write P instead of T. Thus, we assume that $P = (p_1, p_2, \ldots, p_n)$ is a polygonal path.

We will need the following result, which is Lemma 5.8 in Driemel and Har-Peled [17].

Lemma 1 *Let P' be a polygonal path in $\mathbb{R}^d$ with n' vertices and let $\varepsilon > 0$ be a real number. In $O((1/\varepsilon^{2d}) \log^2(1/\varepsilon) n' \log n')$ time, we can construct a data structure of size $O((1/\varepsilon^{2d}) \log^2(1/\varepsilon))$, such that for any query segment $Q = [a, b]$, we can compute, in $O(1)$ time, a real number Δ_Q such that*

$$\delta_F(Q, P') \leq \Delta_Q \leq (1 + \varepsilon) \cdot \delta_F(Q, P').$$

Recall that $\Delta > 0$ and $\varepsilon > 0$ are fixed real numbers. Our data structure is a simplified version of the structure in Section 5.4 of [17]. It consists of the following:

1. A balanced binary search tree storing the n edges of the polygonal path P at its leaves, sorted in the order in which they appear along P.
2. Each node of this tree stores the data structure of Lemma 1 (applied with $\varepsilon/2$) for the subpath of P that is stored in its subtree.

It follows from Lemma 1 that this entire data structure has size $O((1/\varepsilon^{2d}) \log^2(1/\varepsilon) n)$ and can be built in $O((1/\varepsilon^{2d}) \log^2(1/\varepsilon) n \log^2 n)$ time.

We now describe the query algorithm, which is a minor variation of the one in [17]. Consider a query segment $Q = [a, b]$ and two points x and y on P, together with the two edges of P that contain x and y.

Step 1: Use the binary search tree to partition the path $P[x, y]$ into $k = O(\log n)$ subpaths $P[x_i, x_{i+1}]$, $i = 0, 1, 2, \ldots, k-1$, where $x_0 = x$, $x_k = y$, and each of $x_1, x_2, \ldots, x_{k-1}$ is a vertex of $P[x, y]$.

Step 2: Imagine partitioning the segment $Q = [a, b]$ uniformly into subsegments of length $\varepsilon\Delta/3$, except for possibly the last one, which may be shorter. Let Π be the set of vertices of this imaginary partition. For each i with $1 \leq i < k$, compute the set

$$V_i = \{u \in \Pi : |x_i u| \leq 2\Delta\}.$$

Step 3: Construct the directed acyclic graph G with vertex multiset

$$\{a\} \cup V_1 \cup V_2 \cup \cdots \cup V_{k-1} \cup \{b\}.$$

For any i with $1 \leq i \leq k-2$, and any two vertices u in V_i and v in V_{i+1}, (u, v) is a directed edge in G if and only if the point v is on the line segment $[u, b]$. Observe that our data structure stores the data structure of Lemma 1 for the subpath $P[x_i, x_{i+1}]$. We run the query algorithm of this data structure for the query segment $[u, v]$ and store the result, which is a $(1 + \varepsilon/2)$-approximation to $\delta_F([u, v], P[x_i, x_{i+1}])$, as the *weight* of the directed edge (u, v).

For each vertex v in V_1, we add the directed edge (a, v) to G and compute its weight as $\delta_F([a, v], [x, x_1])$. Finally, for each vertex u in V_{k-1}, we add the directed edge (u, b) to G and compute its weight as $\delta_F([u, b], [x_{k-1}, y])$.

Step 4: Use dynamic programming to compute a *bottleneck path* in G from the vertex a to the vertex b, i.e., a path from a to b whose heaviest edge is minimum. Let Δ' be the weight of the heaviest edge on this bottleneck path.

Step 5: If $\Delta' \leq (1 + \varepsilon)\Delta$, then return *true*. Otherwise, return *false*.

We start by analyzing the running time of this query algorithm. Step 1 takes $O(\log n)$ time. The set V_i that is computed in Step 2 is contained in the ball B of radius 2Δ that is centered at x_i. Since any two points in Π have distance at least $\varepsilon\Delta/3$, it follows that V_i has size $O(1/\varepsilon)$. To compute V_i, we first compute, in $O(1)$ time, the part Q' of the segment $Q = [a, b]$ that is contained in B. Then, using the floor function, we enumerate the vertices of $V_i = Q' \cap \Pi$ in $O(|V_i|) = O(1/\varepsilon)$ time. Thus, the total time for Step 2 is $O(k/\varepsilon) = O((\log n)/\varepsilon)$. For each i with $1 \leq i \leq k-2$, the graph G has $O(1/\varepsilon^2)$ edges connecting V_i and V_{i+1}. As a result, the total number of edges in G is $O(k/\varepsilon^2) = O((\log n)/\varepsilon^2)$. Using Lemma 1, the edge weights can be computed in $O(1)$ time per edge. It follows that the total time for Step 3 is $O((\log n)/\varepsilon^2)$. The time for Step 4 is proportional to the sum of the number of vertices and edges of G, which is $O((\log n)/\varepsilon^2)$. Finally, Step 5 takes $O(1)$ time. To conclude, we have shown that the entire query algorithm takes $O((\log n)/\varepsilon^2)$ time.

We now prove the correctness of the query algorithm. Let

$$a = a_0, a_1, a_2, \ldots, a_k = b$$

be the bottleneck path that is computed in Step 4, let Δ' be the weight of the heaviest edge on this path and, for each i with $0 \leq i < k$, let W_i be the weight of (a_i, a_{i+1}) (which is an edge in G). We have

$$\begin{aligned} \delta_F(Q, P[x,y]) &\leq \max_{0 \leq i < k} \delta_F([a_i, a_{i+1}], P[x_i, x_{i+1}]) \\ &\leq \max_{0 \leq i < k} W_i \\ &= \Delta'. \end{aligned}$$

Thus, if the query algorithm returns *true*, then

$$\delta_F(Q, P[x,y]) \leq \Delta' \leq (1+\varepsilon)\Delta.$$

Now assume that $\delta_F(Q, P[x,y]) \leq \Delta$. We will show that the query algorithm returns *true*.

Consider a matching between $Q = [a,b]$ and $P[x,y]$ that realizes $\delta_F(Q, P[x,y])$. For each i with $0 \leq i \leq k$, let b_i be the point on $Q = [a,b]$ that is matched to x_i. Observe that $b_0 = a$, $b_k = b$ and $|x_i b_i| \leq \Delta$ for $0 \leq i \leq k$.

For each i with $1 \leq i < k$, let u_i be the element of $V_i \cap [b_i, b]$ that is closest to b_i. Since, assuming that ε is sufficiently small,

$$|x_i u_i| \leq |x_i b_i| + |b_i u_i| \leq \Delta + \varepsilon\Delta/3 \leq 2\Delta,$$

the point u_i exists. We also define $u_0 = a$ and $u_k = b$.

Consider the path

$$a = u_0, u_1, u_2, \ldots, u_k = b$$

in the graph G. For each i with $0 \leq i < k$, let W'_i be the weight of the edge (u_i, u_{i+1}). We have

$$\begin{aligned} W'_i &\leq (1+\varepsilon/2) \cdot \delta_F([u_i, u_{i+1}], P[x_i, x_{i+1}]) \\ &\leq (1+\varepsilon/2) \left(\delta_F([b_i, b_{i+1}], P[x_i, x_{i+1}]) + \varepsilon\Delta/3\right). \end{aligned}$$

Thus, the value Δ' that is computed in Step 4 satisfies

$$\begin{aligned} \Delta' &\leq \max_{0 \leq i < k} W'_i \\ &\leq (1+\varepsilon/2) \cdot \varepsilon\Delta/3 + (1+\varepsilon/2) \cdot \max_{0 \leq i < k} \delta_F([b_i, b_{i+1}], P[x_i, x_{i+1}]) \\ &= (1+\varepsilon/2) \cdot \varepsilon\Delta/3 + (1+\varepsilon/2) \cdot \delta_F(Q, P[x,y]) \\ &\leq (1+\varepsilon/2) \cdot \varepsilon\Delta/3 + (1+\varepsilon/2)\Delta \\ &\leq (1+\varepsilon)\Delta, \end{aligned}$$

where we assumed in the last inequality that ε is sufficiently small. It follows that the query algorithm returns *true*.

We summarize the result of this section:

Lemma 2 *Let P be a polygonal path in $\mathbb{R}^d$ with n vertices and let $\Delta > 0$ and $\varepsilon > 0$ be real numbers. In $O((1/\varepsilon^{2d}) \log^2(1/\varepsilon) n \log^2 n)$ time, we can construct a data structure of size $O((1/\varepsilon^{2d}) \log^2(1/\varepsilon) n)$, such that for any query consisting of a line segment $Q = [a, b]$ and two points x and y on P (together with the edges of P that contain x and y), we can return, in $O((\log n)/\varepsilon^2)$ time, a Boolean value B that satisfies the following two properties:*

1. *If $B = true$, then $\delta_F(Q, P[x, y]) \leq (1 + \varepsilon)\Delta$.*
2. *If $B = false$, then $\delta_F(Q, P[x, y]) > \Delta$.*

2.2 Queries in Trees

In this section, we extend the data structure of Lemma 2 from polygonal paths to trees. Let T be a tree with n vertices in $\mathbb{R}^d$.

2.2.1 The Path Decomposition

We start by recalling a technique due to Cole and Vishkin [14] that decomposes the tree T into a collection of paths such that any path in T overlaps $O(\log n)$ paths in the decomposition. (A detailed description of this technique can be found in Section 2.3.2 of Narasimhan and Smid [22].)

Fix one vertex of T and call it the root. For any vertex v of T, the *subtree* of v is the set of all vertices u of T such that v is on the path between u and the root. Let $size(v)$ denote the number of vertices in the subtree of v, and let $\ell(v) = \lfloor \log(size(v)) \rfloor$. Thus, $\ell(v)$ is an integer in $\{0, 1, \ldots, \lfloor \log n \rfloor\}$.

For a given integer ℓ, consider a maximal connected subgraph of T consisting of vertices v with $\ell(v) = \ell$. This subgraph is in fact a path which is contained in some root-to-leaf path. Thus, the values $\ell(v)$ induce a partition of the vertex set of V into a collection of paths. Note that no two paths in this partition share a vertex. For any path in the partition, let v be the vertex on this path that is closest to the root (with respect to distances along T). If v is not the root, then we add the parent of v to the path. The resulting collection of paths is called the *path decomposition* $PD(T)$ of the tree T. Any two paths in $PD(T)$ are edge-disjoint and have at most one vertex in common.

For any two points x and y on T, the path $T[x, y]$ overlaps $O(\log n)$ paths in $PD(T)$. More precisely, given x and y, in $O(\log n)$ time, a sequence $v_1, \ldots, v_k$ can be computed, such that

1. $k = O(\log n)$,
2. $v_1 = x$ and $v_k = y$,
3. for each i with $2 \leq i \leq k - 1$, v_i is an endpoint of some path in $PD(T)$ (and, thus, a vertex of T),
4. for each i with $1 \leq i < k$, the path $T[v_i, v_{i+1}]$ is contained in some path in $PD(T)$,

5. the path $T[x, y]$ in T between x and y is equal to the concatenation of the paths $T[v_1, v_2], T[v_2, v_3], \ldots, T[v_{k-1}, v_k]$.

Using this, the following type of query can be answered in $O(\log n)$ time: Given three points x, y, and z on T, together with the edges that contain them, decide if z is on the path $T[x, y]$. The following lemma states that such a query can be answered in $O(1)$ time:

Lemma 3 *Let T be a tree in $\mathbb{R}^d$ with n vertices. In $O(n)$ time, we can construct a data structure of size $O(n)$, such that for any three points x, y, and z on T, together with the edges that contain them, we can decide in $O(1)$ time if z is on the path $T[x, y]$.*

Proof. Fix a vertex of T to be the root, and for each non-leaf vertex, fix a left-to-right order of v's children. Number the leaves of T from left to right as $1, 2, 3, \ldots$ With each node v, store the interval $[L_v, R_v]$, where L_v and R_v are the indices of the leftmost and rightmost leaves in the subtree rooted at v. Observe that for any two nodes v and w, v is in the subtree of w if and only if $I_v \subseteq I_w$. Finally, preprocess T for $O(1)$–time lowest common ancestor queries; see Bender and Farach-Colton [6].

Consider three points x, y, and z, together with the edges e_x, e_y, and e_z that contain them. Let x' be the vertex of e_x that is closest to the root. Define y' and z' similarly with respect to y and z. Compute the lowest common ancestor v of x' and y'. Then z is on the path $T[x, y]$ if and only if z' is in the subtree of v, and x' or y' is in the subtree of z'. ∎

2.2.2 The Data Structure

As before, we fix real numbers $\Delta > 0$ and $\varepsilon > 0$. To construct the data structure, we compute the path decomposition $PD(T)$ of T, which can be done in $O(n)$ time. Then we construct the data structure of Lemma 2 for each path in this decomposition. The preprocessing time and size of this data structure are the same as in Lemma 2.

Consider a query consisting of a line segment $Q = [a, b]$ and two points x and y on T, together with the edges of T that contain x and y. We first use the path decomposition $PD(T)$ to decompose the path $T[x, y]$ into $O(\log n)$ paths. Then we use the binary search tree associated with each such path to decompose it further into $O(\log n)$ subpaths. Thus, the entire path $T[x, y]$ is now decomposed into $k' = O(\log^2 n)$ subpaths. In the rest of the query algorithm, we run Steps 2–5 of the query algorithm in Section 2.1, where the value of k is replaced by k'. The same analysis as in Section 2.1 implies the following result:

Lemma 4 *Let T be a tree in $\mathbb{R}^d$ with n vertices and let $\Delta > 0$ and $\varepsilon > 0$ be real numbers. In $O((1/\varepsilon^{2d}) \log^2(1/\varepsilon) n \log^2 n)$ time, we can construct a data structure of size $O((1/\varepsilon^{2d}) \log^2(1/\varepsilon) n)$, such that for any query consisting of a line segment $Q = [a, b]$ and two points x and y on T (together with the edges of T that contain x and y), we can return, in $O((\log^2 n)/\varepsilon^2)$ time, a Boolean value B that satisfies the following two properties:*

1. *If $B = true$, then $\delta_F(Q, T[x, y]) \leq (1 + \varepsilon)\Delta$.*
2. *If $B = false$, then $\delta_F(Q, T[x, y]) > \Delta$.*

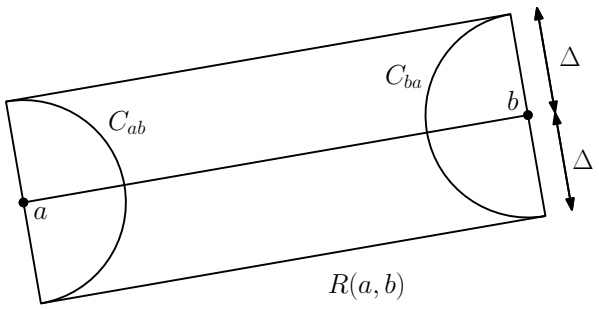

Figure 1: The cylinder $R(a,b)$ and the half-spheres C_{ab} and C_{ba} corresponding to the line segment ab.

---

3 A Generic Matching Algorithm for Query Segments

In this section, we present a generic algorithm that can be used to answer approximate Fréchet matching queries with any query segment Q of length more than 2Δ in any tree T. We start by presenting the main technical lemma that will be used to prove the correctness of our approach. Afterwards, the generic algorithm itself will be presented.

Let T be a tree in $\mathbb{R}^d$ and let $\Delta > 0$ be a real number. We assume that a query consists of a line segment Q with endpoints a and b, which we denote by $Q = [a,b]$. We also assume that $|ab| > 2\Delta$.

Let $R(a,b)$ be the cylinder with axis $[a,b]$ and radius Δ; refer to Figure 1. Let D_a be the ball with center a and radius Δ, and let C_{ab} be the part of the boundary of this ball that is contained in $R(a,b)$. Define D_b and C_{ba} similarly with respect to b.

Assume that there exist two points x and y on T such that $\delta_F(Q, T[x,y]) \leq \Delta$. The following lemma states that we may assume that x is on C_{ab}, y is on C_{ba}, and the "open" path $T(x,y)$ (i.e., the path obtained by removing the endpoints x and y from $T[x,y]$) is disjoint from both C_{ab} and C_{ba}.

Lemma 5 *Let $Q = [a,b]$ be a line segment of length more than 2Δ, and assume that there exist two points x and y on T, such that $\delta_F(Q, T[x,y]) \leq \Delta$. Then there exist two points x' and y' on $T[x,y]$, such that the following are true:*

1. *x' and y' are on the half-spheres C_{ab} and C_{ba}, respectively, and x' is on the path $T[x,y']$.*
2. *The open path $T(x',y')$ is disjoint from $C_{ab} \cup C_{ba}$.*
3. *$\delta_F(Q, T[x',y']) \leq \Delta$.*

Proof. Let $R'(a,b)$ be the union of $R(a,b)$, D_a, and D_b. Since $\delta_F(Q, T[x,y]) \leq \Delta$, (i) the path $T[x,y]$ is completely contained in $R'(a,b)$, (ii) the point x is in D_a, and (iii) the point y is in D_b. Define x' to be the last point on the path $T[x,y]$ that is in D_a, and define y' to

be the first point on $T[x, y]$ that is in D_b. It is clear that the first two claims in the lemma hold. The third claim follows from de Berg *et al.* [15, Lemma 1]. ∎

We now present our generic algorithm that approximately decides if there exist two points x and y on T such that $\delta_F(Q, T[x, y]) \leq \Delta$. The basic idea is to consider all pairs (x', y') of points on T for which the three conditions in Lemma 5 hold.

Fix a real number $\varepsilon > 0$. Consider a query segment $Q = [a, b]$ of length more than 2Δ.

Step 1:

- Compute the set A of intersection points between the tree T and the half-sphere C_{ab}.
- Compute the set B of intersection points between the tree T and the half-sphere C_{ba}.

Step 2: Compute the set $I = \{(x, y) \in A \times B : T(x, y) \cap A = \emptyset \text{ and } T(x, y) \cap B = \emptyset\}$.

Step 3: For each pair (x, y) in I, do the following:

- Compute a Boolean value B_{xy} that satisfies the following two properties:
  - If $B_{xy} = \mathit{true}$, then $\delta_F(Q, T[x, y]) \leq (1 + \varepsilon)\Delta$.
  - If $B_{xy} = \mathit{false}$, then $\delta_F(Q, T[x, y]) > \Delta$.
- If $B_{xy} = \mathit{true}$, then return YES, together with the two points x and y, and terminate the algorithm.

If, at the end of this third step, the algorithm did not terminate yet, then return NO and terminate.

In the following lemma, we analyze the output of this algorithm.

Lemma 6 *Let $Q = [a, b]$ be a line segment of length more than 2Δ and consider the output of the generic algorithm on input Q.*

1. *If the output is YES, then there exist two points x and y on T such that $\delta_F(Q, T[x, y]) \leq (1 + \varepsilon)\Delta$.*
2. *If the output is NO, then for any two points x and y on T, $\delta_F(Q, T[x, y]) > \Delta$.*

Proof. Assume that the output of the algorithm is YES. Consider the two points x and y that come with this output. It follows from the algorithm that

$$\delta_F(Q, T[x, y]) \leq (1 + \varepsilon)\Delta.$$

To prove the second claim, assume there exist two points x and y on T such that $\delta_F(Q, T[x, y]) \leq \Delta$. Let x' and y' be the two points on T that satisfy the three properties in Lemma 5. Then the pair (x', y') is contained in the set I that is computed in Step 2. Consider the Boolean value $B_{x'y'}$ that is computed by the algorithm when it considers this

pair in Step 3. Since $\delta_F(Q, T[x', y']) \leq \Delta$, we have $B_{x'y'} = \mathit{true}$ and, therefore, the algorithm returns YES. ∎

If n denotes the number of vertices of T, then the worst-case running time of the generic algorithm will be $\Omega(n^2)$, because the sets A and B computed in Step 1 may have size $\Theta(n)$ and, thus, the set I computed in Step 2 may have size $\Theta(n^2)$. In the next section, we recall c-packed trees and prove some of their properties. As we will see, if the tree T is c-packed, for some c that is polylogarithmic in n, and each of its edges has length $\Omega(\Delta)$, the running time of the generic algorithm will be polylogarithmic in n.

4 c-Packed Trees and μ-Simplifications

The notion of a tree being c-packed was introduced by Driemel *et al.* [18]. We start by recalling the definition.

Definition 1 *Let c be a positive real number and let T be a tree in $\mathbb{R}^d$. We say that T is* c-packed *if for any $\rho > 0$ and any ball B of radius ρ, the total length of $B \cap T$ is at most $c\rho$.*

The number of edges of a c-packed tree T that intersect a ball B can be as large as $\Omega(n)$. The next lemma gives an upper bound on this number for the case when the length of each edge of T is at least proportional to the radius of B.

Lemma 7 *Let T be a c-packed tree, let $\Delta > 0$ and $c' \geq 1$ be real numbers, and assume that each edge of T has length at least Δ/c'. Then for any ball B of radius Δ, the number of edges of T that intersect B is $O(cc')$.*

Proof. Let B' be the ball of radius $(1 + 1/c')\Delta$ that has the same center as B. Let e be an edge of T that intersects B. Since $|e| \geq \Delta/c'$, at least Δ/c' of the length of e is contained in B'. If k is the number of edges of T that intersect B, then $k(\Delta/c') \leq c(1 + 1/c')\Delta$, which implies that $k \leq c(c' + 1) = O(cc')$. ∎

In Section 5.3, we consider polygonal paths P that are c-packed, but do not have the property that each edge has length $\Omega(\Delta)$. Since Lemma 7 does not hold for such a path, we would like to simplify P, resulting in a path P', such that P' is $O(c)$-packed, each of its edges has length $\Omega(\Delta)$, and P' is close to P with respect to the Fréchet distance. Driemel *et al.* [18] gave a simple algorithm that computes such a path P'. Before we state their result, we formally define the notion of a simplification.

Definition 2 *Let $P = (p_1, p_2, \ldots, p_n)$ be a polygonal path in $\mathbb{R}^d$ and let $\mu > 0$ be a real number. A* μ-simplification *of P is a polygonal path $P' = (p_{i_1}, p_{i_2}, \ldots, p_{i_k})$ such that*

1. *$1 = i_1 < i_2 < \ldots < i_k = n$ and*
2. *each edge of P', except possibly the last one, has length at least μ.*

Lemma 8 (Driemel *et al.* [18]) *The following are true:*

1. *In $O(n)$ time, a μ-simplification P' of P can be computed such that $\delta_F(P, P') \leq \mu$.*
2. *If the polygonal path P is c-packed, then this μ-simplification P' is $(6c)$-packed.*

Proof. A proof of the first claim can be found in Section 2.3 of [18]. The second claim is Lemma 4.3 in [18]. ∎

We need one more result:

Lemma 9 *Let T be a c-packed tree with n vertices in $\mathbb{R}^d$, let $\Delta > 0$ and $c' \geq 1$ be real numbers, and assume that each edge of T has length at least Δ/c'. In $O(cn \log n)$ time, we can construct a data structure of size $O(cn)$ such that for any query sphere S of radius Δ, we can report all intersection points between T and S in $O(\log n + c^2 c')$ time.*

Proof. Let B be the ball whose boundary is S. By Lemma 7, the number of edges of T that intersect B is $O(cc')$. Since each such edge intersects S at most twice, the number of intersection points between T and S is $O(cc')$ as well.

Driemel *et al.* [18] have shown that T is a $(2c)$-low density scene. De Berg and Streppel [16] have shown how this type of convex range queries on low density scenes can be handled efficiently using a binary space partition (Theorem 4.1 in [16]). ∎

5 Matching Queries with Segments in Polygonal Paths

In this section, we assume that the tree T is a polygonal path and write P instead of T. Thus, we assume that $P = (p_1, p_2, \ldots, p_n)$ is a polygonal path in $\mathbb{R}^d$. If x and y are two points on P, then we write $x \leq_P y$ if x is on the subpath $P[p_1, y]$.

We fix a real number $\Delta > 0$ and consider queries of the following type: Given a query segment $Q = [a, b]$ of length more than 2Δ, decide if there exist two points x and y on P such that $x \leq_P y$ and $\delta_F(Q, P[x, y]) \leq \Delta$.

We choose positive real numbers c, c', and ε. Throughout this section, we assume that P is c-packed. We also start by assuming that each edge of P, except possibly the last one, has length at least Δ/c'. In Section 5.3, we will show how to remove the latter assumption.

5.1 Preprocessing

We construct the following two data structures:

Approximate subpath Fréchet distance structure: The data structure of Lemma 2 for the polygonal path P. We denote this structure by $ASFD(P, \Delta)$.

Half-sphere intersection structure: The data structure of Lemma 9 for the polygonal path P. We denote this structure by $HSI(P, \Delta)$.

By Lemmas 2 and 9, the entire preprocessing takes

$$O\left(\left((1/\varepsilon^{2d})\log^2(1/\varepsilon)\right) n \log^2 n + cn \log n\right)$$

time and produces a data structure of size

$$O\left(\left(c + (1/\varepsilon^{2d})\log^2(1/\varepsilon)\right) n\right).$$

5.2 The Query Algorithm

Let $Q = [a, b]$ be a query line segment of length more than 2Δ. Recall that we assume that the polygonal path P is c-packed and each edge of P, except possibly the last one, has length at least Δ/c'.

We show how our data structures can be used to implement the three steps of the generic algorithm of Section 3.

Step 1: The data structure $HSI(P, \Delta)$ allows us to compute the set A of all intersection points between P and the half-sphere C_{ab} in $O(\log n + c^2c')$ time. In the same time bound, we obtain the set B of all intersection points between P and the half-sphere C_{ba}. Observe that, by Lemma 7, both A and B have size $O(cc')$.

Step 2: We sort the points of $A \cup B$ in the order in which they occur along the path P. By scanning the sorted order, we obtain the set

$$I = \{(x, y) \in A \times B : x \leq_P y, P(x, y) \cap A = \emptyset \text{ and } P(x, y) \cap B = \emptyset\}.$$

Since $|A| + |B| = O(cc')$, this part of the query algorithm takes $O(cc' \log(cc'))$ time.

Step 3: For each pair (x, y) in the set I, we use the data structure $ASFD(P, \Delta)$ to compute a Boolean value B_{xy}. If $B_{xy} = true$, then return YES, together with the two points x and y, and terminate the algorithm.

If, at the end of this third step, the algorithm did not terminate yet, then return NO and terminate.

By Lemma 2, and since the size of I is $O(cc')$, this part of the query algorithm takes $O((cc'/\varepsilon^2)\log n)$ time.

This concludes the description of the query algorithm. It is clear that these three steps correctly implement the generic algorithm of Section 3.

Lemma 10 *The query algorithm takes*

$$O\left(\left(cc'/\varepsilon^2\right)\log n + c^2c' + cc'\log(cc')\right)$$

time and correctly implements the generic algorithm of Section 3. Thus, the two claims in Lemma 6 hold for the output of this algorithm.

Note that this result assumes that each edge of P, except possibly the last one, has length at least Δ/c'. In the next subsection, we remove this assumption.

5.3 General c-Packed Paths

Let $P = (p_1, p_2, \ldots, p_n)$ be a polygonal path and assume that P is c-packed for some real number $c > 0$. As before, we choose real numbers Δ, ε, and c'.

Let $\mu = \Delta/c'$. In $O(n)$ time, we compute a μ-simplification P' of P; see Lemma 8. Then we run the preprocessing algorithm of Section 5.1 on P'. For any given query segment $Q = [a, b]$ of length more than 2Δ, we run the query algorithm of Section 5.2 on the data structure for P'.

Assume the output of the query algorithm is YES. By Lemma 6, there exist two points x' and y' on P' such that $x' \leq_{P'} y'$ and $\delta_F(Q, P'[x', y']) \leq (1 + \varepsilon)\Delta$. Since, by Lemma 8, $\delta_F(P, P') \leq \mu = \Delta/c'$, it follows that there exist two points x and y on P such that $x \leq_P y$ and

$$\delta_F(Q, P[x, y]) \leq (1 + \varepsilon)\Delta + \Delta/c' = (1 + \varepsilon + 1/c')\Delta. \tag{1}$$

On the other hand, if the output of the query algorithm is NO, then we have, by Lemma 6, $\delta_F(Q, P'[x', y']) > \Delta$ for any two points x' and y' on P' with $x' \leq_{P'} y'$. Therefore, we have

$$\delta_F(Q, P[x, y]) > \Delta - \mu = (1 - 1/c')\Delta \tag{2}$$

for any two points x and y on P with $x \leq_P y$.

By taking $c' = 1/\varepsilon$ and defining $\Delta_0 = (1 - 1/c')\Delta$, the right-hand side in (1) becomes $\Delta_0(1 + O(\varepsilon))$, whereas the right-hand side in (2) becomes Δ_0. Thus, by replacing ε in the entire construction by ε/c'', for some sufficiently large constant c'', we have proved the following result[2].

Theorem 1 *Let P be a polygonal path in $\mathbb{R}^d$ with n vertices, let c and Δ be positive real numbers, and assume that P is c-packed. For any $\varepsilon > 0$, we can construct a data structure of size*

$$O\left(\left(c + (1/\varepsilon^{2d}) \log^2(1/\varepsilon)\right) n\right)$$

in

$$O\left(\left((1/\varepsilon^{2d}) \log^2(1/\varepsilon)\right) n \log^2 n + cn \log n\right)$$

time. Given any segment Q of length more than 2Δ, the query algorithm corresponding to this data structure takes

$$O\left(\left(c/\varepsilon^3\right) \log n + c^2/\varepsilon + (c/\varepsilon) \log(c/\varepsilon)\right)$$

time and outputs either YES or NO.

1. *If the output is YES, then there exist two points x and y on P with $x \leq_P y$ such that $\delta_F(Q, P[x, y]) \leq (1 + \varepsilon)\Delta$.*
2. *If the output is NO, then $\delta_F(Q, P[x, y]) > \Delta$ for any two points x and y on P with $x \leq_P y$.*

[2]In the theorem, we rename Δ_0 as Δ.

6 Matching Queries with Segments in Trees

Let T be a tree with n vertices in $\mathbb{R}^d$. We fix a real number $\Delta > 0$ and consider queries of the following type: Given a query segment $Q = [a, b]$ of length more than 2Δ, decide if there exist two points x and y on T such that $\delta_F(Q, T[x, y]) \leq \Delta$.

We choose positive real numbers c, c', and ε, and assume that T is c-packed and each edge of T has length at least Δ/c'.

We construct the data structure $ASFD(T, \Delta)$ of Lemma 4, the data structure $B(T)$ of Lemma 3, and the data structure $HSI(T, \Delta)$ of Lemma 9 for the tree T.

Consider a query segment $Q = [a, b]$ of length more than 2Δ. We show how these data structures can be used to implement the three steps of the generic algorithm of Section 3.

Step 1: In exactly the same way as in Section 5.2, we compute, in $O(\log n + c^2c')$ time, the sets A and B of intersection points between T and the half-spheres C_{ab} and C_{ba}, respectively. Recall that both A and B have size $O(cc')$.

Step 2: For each x in A and each y in B, we do the following: For each point z in $(A \cup B) \setminus \{x, y\}$, use the data structure $B(T)$ to decide if z is on the path $T[x, y]$. If this is not the case for all such z, then we add the pair (x, y) to an initially empty set.

At the end of this step, we have computed the set

$$I = \{(x, y) \in A \times B : T(x, y) \cap A = \emptyset \text{ and } T(x, y) \cap B = \emptyset\}.$$

It follows from Lemma 3 that the total time for this step is $O((cc')^3)$. Observe that the size of the set I is $O((cc')^2)$.

Step 3: This step is the same as in Section 5.2, except that we use the data structure $ASFD(T, \Delta)$. Since the size of I is $O((cc')^2)$, the total time for this step is $O((cc'/\varepsilon)^2 \log^2 n)$.

This concludes the description of the query algorithm. Observe that it correctly implements the generic algorithm of Section 3. Therefore, we have proved the following result.

Theorem 2 *Let T be a tree with n vertices in $\mathbb{R}^d$, and let c, c', and Δ be positive real numbers. Assume that T is c-packed and each of its edges has length at least Δ/c'. For any $\varepsilon > 0$, we can construct a data structure of size*

$$O\left(\left(c + (1/\varepsilon^{2d}) \log^2(1/\varepsilon)\right) n\right)$$

in

$$O\left(\left((1/\varepsilon^{2d}) \log^2(1/\varepsilon)\right) n \log^2 n + cn \log n\right)$$

time. Given any segment Q of length more than 2Δ, the query algorithm corresponding to this data structure takes

$$O\left((cc'/\varepsilon)^2) \log^2 n + (cc')^3\right)$$

time and outputs either YES or NO.

1. *If the output is YES, then there exist two points x and y on T such that $\delta_F(Q, T[x, y]) \leq (1 + \varepsilon)\Delta$.*

2. If the output is NO, then $\delta_F(Q, T[x, y]) > \Delta$ for any two points x and y on T.

Remark 1 *If we do not perform Step 2, then the result in Theorem 2 still holds, but without the term $(cc')^3$ in the query time. The advantage of including Step 2 is that it allows us to count all minimal paths $T[x, y]$ whose Fréchet distance to Q is (approximately) at most Δ.*

Unfortunately, the simplification technique of Section 4 cannot be used to remove the assumption that each edge of T has length at least Δ/c': The number of paths in the path decomposition $PD(T)$ of the tree T can be $\Omega(n)$. Therefore, if we apply the simplification technique to each such path, as we did in Section 5.3, we may get $\Omega(n)$ simplified paths, each of which contains one edge of length less than $\mu = \Delta/c'$. As a result, the sets A and B that are computed in Step 1 of the query algorithm may have linear size.

7 Querying with a Polygonal Path

Until now, we have considered querying polygonal paths or trees with a line segment. In this section, we generalize our results to queries Q consisting of a polygonal path. Unfortunately, the approximation factor increases from $1+\varepsilon$ to $3(1+\varepsilon)$, and we need the requirement that each edge of Q has length more than 5Δ.

Let T be a tree in $\mathbb{R}^d$ and let $\Delta > 0$ be a real number. We consider polygonal query paths $Q = (q_1, q_2, \ldots, q_m)$, all of whose edges have length more than 5Δ. As before, we fix a real number $\varepsilon > 0$. The following lemma (which only needs the requirement that each edge of Q has length more than 2Δ) generalizes Lemma 5.

Lemma 11 *Let $Q = (q_1, q_2, \ldots, q_m)$ be a polygonal path, all of whose edges have length more than 2Δ. Assume there exist two points x and y on T, such that $\delta_F(Q, T[x, y]) \leq \Delta$. Then there exist points $x'_1, y'_1, \ldots, x'_{m-1}, y'_{m-1}$ on $T[x, y]$, such that the following are true:*

1. *By traversing the path $T[x, y]$, we visit the points $x'_1, y'_1, \ldots, x'_{m-1}, y'_{m-1}$ in this order.*
2. *For each integer i with $1 \leq i < m$,*
   (a) *x'_i and y'_i are on the half-spheres $C_{q_i q_{i+1}}$ and $C_{q_{i+1} q_i}$, respectively,*
   (b) *the open path $T(x'_i, y'_i)$ is disjoint from $C_{q_i q_{i+1}} \cup C_{q_{i+1} q_i}$,*
   (c) *$\delta_F([q_i, q_{i+1}], T[x'_i, y'_i]) \leq \Delta$.*
3. *For each integer i with $1 \leq i < m-1$, the path $T[y'_i, x'_{i+1}]$ is completely contained in the ball with center q_{i+1} and radius 3Δ.*

Proof. Consider a matching between Q and $T[x, y]$ that realizes $\delta_F(Q, T[x, y])$. For each i with $1 \leq i \leq m$, let x_i be the point on $T[x, y]$ that is matched to q_i. Observe that $x_1 = x$, $x_m = y$, and $|q_i x_i| \leq \Delta$ for $1 \leq i \leq m$.

Let i be any index such that $1 \leq i < m$. It is obvious that $\delta_F([q_i, q_{i+1}], T[x_i, x_{i+1}]) \leq \Delta$. Therefore, we can apply Lemma 5 to the line segment $[q_i, q_{i+1}]$ and the two points x_i and

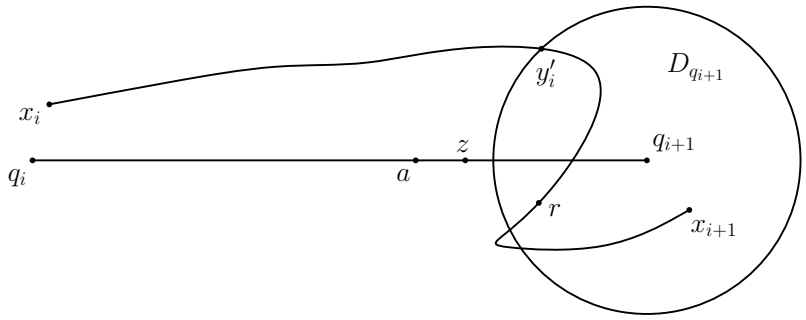

Figure 2: Illustrating the proof of the third claim in Lemma 11. The matching that realizes $\delta_F(Q, T[x, y])$ matches a with y'_i and z with r.

---

x_{i+1} on T. We obtain two points x'_i and y'_i on the path $T[x_i, x_{i+1}]$ such that the first and second claims hold.

It remains to prove the third claim. Let i be any index such that $1 \leq i < m-1$. Recall from the proof of Lemma 5 that y'_i is the first point on the path $T[x_i, x_{i+1}]$ that is in the ball $D_{q_{i+1}}$, whereas x'_{i+1} is the last point on $T[x_{i+1}, x_{i+2}]$ that is in $D_{q_{i+1}}$. We have to show that $|q_{i+1}r| \leq 3\Delta$ for each point r on the path $T[y'_i, x'_{i+1}]$. We will prove this for the case when r is on $T[y'_i, x_{i+1}]$; the other case can be proved in a symmetric way.

In the following, refer to Figure 2. Consider an arbitrary point r on $T[y'_i, x_{i+1}]$. Let a be the point on $[q_i, q_{i+1}]$ that is matched to y'_i and let z be the point on $[a, q_{i+1}]$ that is matched to r. We have

$$\begin{aligned} |q_{i+1}r| &\leq |q_{i+1}z| + |zr| \\ &\leq |q_{i+1}a| + |zr| \\ &\leq |q_{i+1}y'_i| + |y'_i a| + |zr| \\ &\leq 3\Delta. \end{aligned}$$

This completes the proof of the lemma. ∎

The following lemma is a sort of converse to Lemma 11. Again, this lemma only needs the requirement that each edge of Q has length more than 2Δ. We leave the easy proof to the reader.

Lemma 12 *Let $Q = (q_1, q_2, \ldots, q_m)$ be a polygonal path, all of whose edges have length more than 2Δ. Assume there exist points $x_1, y_1, \ldots, x_{m-1}, y_{m-1}$ on T, such that the following are true:*

1. *By traversing the path $T[x_1, y_{m-1}]$, we visit the points $x_1, y_1, \ldots, x_{m-1}, y_{m-1}$ in this order.*
2. *For each integer i with $1 \leq i < m$, $\delta_F([q_i, q_{i+1}], T[x_i, y_i]) \leq (1+\varepsilon)\Delta$.*

3. *For each integer i with $1 \leq i < m-1$, the path $T[y_i, x_{i+1}]$ is completely contained in the ball with center q_{i+1} and radius $3(1+\varepsilon)\Delta$.*

Then, $\delta_F(Q, T[x_1, y_{m-1}]) \leq 3(1+\varepsilon)\Delta$.

Our algorithm will run the query algorithms of the previous sections separately on each edge of the query path Q. Afterwards, we check if the partial paths in the tree T obtained for the edges of Q can be combined into one global path that is close to the entire path Q with respect to the Fréchet distance. The following lemma will imply that this combining step leads to an efficient algorithm. We remark that this is where we need the assumption that each edge of Q has length more than 5Δ.

Lemma 13 *Assume that $0 < \varepsilon \leq 1/3$. Let $P = (p_1, p_2, \ldots, p_n)$ and $Q = (q_1, q_2, \ldots, q_m)$ be polygonal paths and assume that each edge of Q has length more than 5Δ. Let i be an index with $1 \leq i < m-1$ and assume that there exists a point y on P, such that y is on the half-sphere $C_{q_{i+1}q_i}$. Then there exists at most one pair (x', y') of points on P such that*

1. *$y \leq_P x' \leq_P y'$,*
2. *x' and y' are on the half-spheres $C_{q_{i+1}q_{i+2}}$ and $C_{q_{i+2}q_{i+1}}$, respectively,*
3. *the open path $P(x', y')$ is disjoint from $C_{q_{i+1}q_{i+2}} \cup C_{q_{i+2}q_{i+1}}$,*
4. *the path $P[y, x']$ is completely contained in the ball with center q_{i+1} and radius $3(1+\varepsilon)\Delta$.*

Proof. Assume that there exist two such pairs (x', y') and (x'', y'') of points on P. It follows from the second and third items that the paths $P[x', y']$ and $P[x'', y'']$ do not overlap. Thus, we may assume without loss of generality that $y' \leq_P x''$. We have

$$5\Delta < |q_{i+1}q_{i+2}| \leq |q_{i+1}y'| + |y'q_{i+2}| = |q_{i+1}y'| + \Delta,$$

and, therefore,

$$|q_{i+1}y'| > 4\Delta \geq 3(1+\varepsilon)\Delta.$$

Since y' is on the path $P[y, x'']$, it follows that the path $P[y, x'']$ is not completely contained in the ball with center q_{i+1} and radius $3(1+\varepsilon)\Delta$. Thus, the pair (x'', y'') does not satisfy the fourth item in the lemma. This is a contradiction, because we assumed that (x'', y'') satisfies all four items in the lemma. ∎

7.1 Querying a Polygonal Path with a Polygonal Path

Assume that the tree T is a polygonal path. As before, we write P instead of T. Thus, we assume that $P = (p_1, p_2, \ldots, p_n)$ is a polygonal path in $\mathbb{R}^d$.

We choose positive real numbers Δ, ε, c, and c', and assume that P is c-packed. As before, we start by assuming that each edge of P, except possibly the last one, has length at least Δ/c'.

Preprocessing: We run the preprocessing algorithm of Section 5.1, resulting in the structures $HSI(P, \Delta)$ and $ASFD(P, \Delta)$. We also construct the data structure of Lemma 2, with the value Δ replaced by 3Δ; we denote this structure by $ASFD(P, 3\Delta)$.

The preprocessing algorithm takes

$$O\left(\left((1/\varepsilon^{2d})\log^2(1/\varepsilon)\right) n \log^2 n + cn \log n\right)$$

time and produces a data structure of size

$$O\left(\left(c + (1/\varepsilon^{2d})\log^2(1/\varepsilon)\right) n\right).$$

Observe that $ASFD(P, 3\Delta)$ can be used to approximately decide if a subpath of P is contained in a ball of radius 3Δ: Indeed, consider a point q in $\mathbb{R}^d$ and two points x and y on P. The subpath $P[x, y]$ is contained in the ball with center q and radius 3Δ if and only if the Fréchet distance between the line segment $[q, q]$ and the subpath $P[x, y]$ is at most 3Δ. Therefore, we query $ASFD(P, 3\Delta)$ with the line segment $[q, q]$ and the points x and y. If the query algorithm returns *true*, then, by Lemma 2, $P[x, y]$ is contained in the ball with center q and radius $3(1 + \varepsilon)\Delta$. Otherwise, $P[x, y]$ is not completely contained in the ball with center q and radius 3Δ.

Answering a query: Consider a query path $Q = (q_1, q_2, \ldots, q_m)$ all of whose edges have length more than 5Δ.

For each integer i with $1 \leq i < m$, we run Steps 1 and 2 of the query algorithm of Section 5.2 with the segment $[q_i, q_{i+1}]$. This gives the set I_i of all pairs (x, y) of points such that

- $x \leq_P y$,
- x is an intersection point between P and the half-sphere $C_{q_i q_{i+1}}$,
- y is an intersection point between P and the half-sphere $C_{q_{i+1} q_i}$,
- the open path $P(x, y)$ is disjoint from $C_{q_i q_{i+1}} \cup C_{q_{i+1} q_i}$.

Next, we modify Step 3 of the algorithm in Section 5.2, in the following way: We consider all pairs in I_i. For each such pair (x, y), we query $ASFD(P, \Delta)$ with the line segment $[q_i, q_{i+1}]$ and the subpath $P[x, y]$. If the Boolean value returned by the query algorithm is *true*, then we insert the pair (x, y) into an initially empty set J_i.

After these steps have been performed for all values of i, we proceed as follows.

- Initialize $K_1 = J_1$.
- For $i = 1, 2, \ldots, m - 2$, do the following:
  - At this moment, we have the set K_i.

- Initialize $K_{i+1} = \emptyset$.
- For each pair (x, y) in K_i, do the following: Consider all pairs in J_{i+1}. For each such pair (x', y'), run the query algorithm of $ASFD(P, 3\Delta)$ with the segment $[q_{i+1}, q_{i+1}]$ and the points y and x'. (Thus, we approximately decide if the subpath $P[y, x']$ is contained in the ball with center q_{i+1} and radius 3Δ.) If the query algorithm returns *true* and $y \leq_P x'$, then we insert the pair (x, y') into the set K_{i+1}.

At the end of this algorithm, we have obtained a set K_{m-1}. If this set is non-empty, then we return YES; otherwise, we return NO.

We first prove the correctness of this algorithm. Assume that the output of the algorithm is YES. Then it follows from Lemmas 2 and 12 that there exist two points x and y on P such that $x \leq_P y$ and $\delta_F(Q, P[x, y]) \leq 3(1 + \varepsilon)\Delta$.

On the other hand, assume there exist two points x and y on P such that $x \leq_P y$ and $\delta_F(Q, P[x, y]) \leq \Delta$. Consider the points $x'_1, y'_1, \ldots, x'_{m-1}, y'_{m-1}$ that satisfy the three properties of Lemma 11. We observe that, for each i with $1 \leq i < m$, the pair (x'_i, y'_i) is contained in the set J_i. Using the third property of Lemma 11, it then follows that for each i with $1 \leq i < m$, the pair (x'_1, y'_i) is in the set K_i. In particular, at the end of the query algorithm, the pair (x'_1, y'_{m-1}) is in the set K_{m-1}. Thus, since this set is non-empty, the query algorithm returns YES.

We next analyze the running time of the query algorithm. By Lemma 7, each of the sets $I_1, I_2, \ldots, I_{m-1}$ has size $O(cc')$. Since $J_i \subseteq I_i$, each of the sets $J_1, J_2, \ldots, J_{m-1}$ has size $O(cc')$ as well.

Let i be an integer with $1 \leq i \leq m - 2$. We claim that $|K_{i+1}| \leq |K_i|$. To prove this, let (x, y) be an arbitrary element in K_i. If (x', y') is an element of J_{i+1} for which our algorithm inserts the pair (x, y') into K_{i+1}, then the points y, x', and y' satisfy the properties in Lemma 13. Thus, Lemma 13 implies that the pair (x, y) of K_i contributes at most one pair (x, y') to K_{i+1}. This proves that $|K_{i+1}| \leq |K_i|$.

For each integer i with $1 \leq i \leq m - 1$, we thus have

$$|K_i| \leq |K_{i-1}| \leq |K_{i-2}| \leq \ldots \leq |K_1| = |J_1| = O(cc').$$

The total query time is the sum of

- the total time for Steps 1 and 2, which is

$$O\left(m\left(\log n + c^2c' + cc'\log(cc')\right)\right),$$

- the total time for the modified Step 3, which is

$$O\left(\sum_{i=1}^{m-1} |I_i|(\log n)/\varepsilon^2\right) = O\left((cc'/\varepsilon^2)m\log n\right),$$

- the total time for the construction of the sets $K_1, K_2, \ldots, K_{m-1}$, which is

$$O\left(\sum_{i=1}^{m-2} |K_i| \cdot |J_{i+1}|(\log n)/\varepsilon^2\right) = O\left((cc'/\varepsilon)^2 m \log n\right).$$

Thus, assuming that P is c-packed and each of its edges has length at least Δ/c', we obtain a data structure with

$$O\left(\left((1/\varepsilon^{2d})\log^2(1/\varepsilon)\right) n \log^2 n + cn \log n\right)$$

preprocessing time,

$$O\left(\left(c + (1/\varepsilon^{2d})\log^2(1/\varepsilon)\right) n\right)$$

size, and

$$O\left((cc'/\varepsilon)^2 m \log n\right)$$

query time.

To remove the assumption that each edge of P has length at least Δ/c', we first compute a μ-simplification P' of P, with $\mu = \Delta/c'$, as we did in Section 5.3. Then we build the above data structure for the simplified path P'. By choosing c' to be proportional to $1/\varepsilon$, for any given $\varepsilon > 0$, we obtain the following result:

Theorem 3 *Let P be a polygonal path in $\mathbb{R}^d$ with n vertices, let c and Δ be positive real numbers, and assume that P is c-packed. For any $\varepsilon > 0$, we can construct a data structure of size*

$$O\left(\left(c + (1/\varepsilon^{2d})\log^2(1/\varepsilon)\right) n\right)$$

in

$$O\left(\left((1/\varepsilon^{2d})\log^2(1/\varepsilon)\right) n \log^2 n + cn \log n\right)$$

time. Given any polygonal path Q with m vertices, all of whose edges have length more than 5Δ, the query algorithm corresponding to this data structure takes

$$O\left((c^2/\varepsilon^4) m \log n\right)$$

time and outputs either YES or NO.

1. *If the output is YES, then there exist two points x and y on P with $x \leq_P y$ such that $\delta_F(Q, P[x, y]) \leq 3(1 + \varepsilon)\Delta$.*
2. *If the output is NO, then $\delta_F(Q, P[x, y]) > \Delta$ for any two points x and y on P with $x \leq_P y$.*

7.2 Querying a Tree with a Polygonal Path

We finally consider a tree T with n vertices. Let Δ, ε, c, and c' be positive real numbers, and assume that T is c-packed and each of its edges has length at least Δ/c'.

We run the preprocessing algorithm of Section 6, resulting in the structures $HSI(T, \Delta)$, $ASFD(T, \Delta)$, and $B(T)$. We also construct the data structure of Lemma 4, with the value Δ replaced by 3Δ; we denote this structure by $ASFD(T, 3\Delta)$.

The entire preprocessing algorithm takes

$$O\left(\left((1/\varepsilon^{2d})\log^2(1/\varepsilon)\right) n \log^2 n + cn \log n\right)$$

time and the resulting data structure has size

$$O\left(\left(c + (1/\varepsilon^{2d})\log^2(1/\varepsilon)\right) n\right).$$

Let $Q = (q_1, q_2, \ldots, q_m)$ be a query path all of whose edges have length more than 5Δ. For each integer i with $1 \leq i < m$, we run Steps 1 and 2 of the query algorithm of Section 6 with the segment $[q_i, q_{i+1}]$. This gives the set I_i of all pairs (x, y) of points such that

- x is an intersection point between T and the half-sphere $C_{q_i q_{i+1}}$,
- y is an intersection point between T and the half-sphere $C_{q_{i+1} q_i}$,
- the open path $T(x, y)$ is disjoint from $C_{q_i q_{i+1}} \cup C_{q_{i+1} q_i}$.

We modify Step 3 of the algorithm in Section 6, again with the segment $[q_i, q_{i+1}]$, as follows. We consider all pairs in I_i. For each such pair (x, y), we query the data structure $ASFD(T, \Delta)$ with the line segment $[q_i, q_{i+1}]$. If the Boolean value returned is *true*, then we insert the pair (x, y) into an initially empty set J_i.

After these steps have been performed for all values of i, we proceed as follows.

- Initialize $K_1 = J_1$.
- For $i = 1, 2, \ldots, m-2$, do the following:
  - At this moment, we have the set K_i.
  - Initialize $K_{i+1} = \emptyset$.
  - For each pair (x, y) in K_i, do the following: Consider all pairs in J_{i+1}. For each such pair (x', y'), run the query algorithm of $ASFD(T, 3\Delta)$ with the segment $[q_{i+1}, q_{i+1}]$ and the points y and x'. If (i) the query algorithm returns *true*, (ii) y is on the path $T[x, x']$ (which we decide using the structure $B(T)$), and (iii) x' is on the path $T[y, y']$ (which we again decide using $B(T)$), then we insert the pair (x, y') into the set K_{i+1}. (It follows from Lemma 13 that (x, y') was not in K_{i+1} yet.)

At the end of this algorithm, we have obtained a set K_{m-1}. If this set is non-empty, then we return YES; otherwise, we return NO.

The correctness proof of our query algorithm is the same as in Section 7.1. To analyze the query time, first observe that each of the sets I_i and J_i has size $O((cc')^2)$. The set K_i contains pairs (x, y) of points on T with x on $C_{q_1 q_2}$ and y on $C_{q_{i+1} q_i}$. Therefore, the size of K_i is $O((cc')^2)$ as well. An analysis similar to the one in Section 7.1 shows that the total query time is

$$O\left(\left((cc')^4/\varepsilon^2\right) m \log^2 n\right).$$

We have proved our final result:

Theorem 4 *Let T be a tree with n vertices in $\mathbb{R}^d$, and let Δ, c, and c' be positive real numbers. Assume that T is c-packed and each of its edges has length at least Δ/c'. For any $\varepsilon > 0$, we can construct a data structure of size*

$$O\left(\left(c + (1/\varepsilon^{2d}) \log^2(1/\varepsilon)\right) n\right)$$

in

$$O\left(\left((1/\varepsilon^{2d}) \log^2(1/\varepsilon)\right) n \log^2 n + cn \log n\right)$$

time. Given any polygonal path Q with m vertices, all of whose edges have length more than 5Δ, the query algorithm corresponding to this data structure takes

$$O\left(\left((cc')^4/\varepsilon^2\right) m \log^2 n\right)$$

time and outputs either YES or NO.

1. *If the output is YES, then there exist two points x and y on T such that $\delta_F(Q, T[x, y]) \leq 3(1 + \varepsilon)\Delta$.*
2. *If the output is NO, then $\delta_F(Q, T[x, y]) > \Delta$ for any two points x and y on T.*

8 Concluding Remarks

We have presented data structures that store, for a fixed real number $\Delta > 0$, a geometric tree T such that, for any given polygonal query path Q, we can approximately decide if the tree contains a path whose Fréchet distance to Q is at most Δ. We obtained good bounds on the preprocessing time, size, and query time for trees that are c-packed, for some small value of c, and for queries Q in which each edge has length $\Omega(\Delta)$. For general trees, we also need that each of its edges has length $\Omega(\Delta)$; for the case when T is a polygonal path, the latter requirement is not needed. We leave as an open problem to remove this requirement for general trees.

Our approximation factors are $1+\varepsilon$ if Q is a query segment, and $3(1+\varepsilon)$ if Q is a polygonal path. We leave as an open problem to improve the approximation factor for polygonal query paths.

All our results assume that the value Δ is fixed. It would be interesting to obtain an efficient data structure for which Δ is part of the query.

Acknowledgment

We thank an anonymous referee for the conference version [20] of this paper for pointing out reference [17] to us. We also thank the anonymous referees for this journal paper for their constructive remarks that helped improve the presentation. In particular, we thank the referee who suggested to us the algorithm in Section 2, which improved our query times by a factor of $\log \log n$, as well as the algorithm in Lemma 9, which allowed us to prove all our results in $\mathbb{R}^d$.

References

[1] P. K. Agarwal, R. B. Avraham, H. Kaplan, and M. Sharir. Computing the discrete Fréchet distance in subquadratic time. *SIAM Journal on Computing*, 43:429–449, 2014.

[2] H. Alt. The computational geometry of comparing shapes. In *Efficient Algorithms, Essays Dedicated to Kurt Mehlhorn on the Occasion of His 60th Birthday*, pages 235–248. Springer, 2009.

[3] H. Alt, A. Efrat, G. Rote, and C. Wenk. Matching planar maps. *Journal of Algorithms*, 49(2):262–283, 2003.

[4] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications,*, 5:75–91, 1995.

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

[6] M. A. Bender and M. Farach-Colton. The LCA problem revisited. In *Proceedings of the 4th Latin American Symposium on Theoretical Informatics*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94, Berlin, 2000. Springer-Verlag.

[7] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On map-matching vehicle tracking data. In *31st Conference on Very Large Data Bases (VLDB)*, pages 853–864. VLDB Endowment, 2005.

[8] K. Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *Proceedings of the 55th IEEE Symposium on Foundations of Computer Science*, 2014.

[9] K. Bringmann and M. Künnemann. Improved approximation for Fréchet distance on c-packed curves matching conditional lower bounds. *CoRR*, abs/1408.1340, 2014.

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

[11] K. Buchin, M. Buchin, C. Knauer, G. Rote, and C. Wenk. How difficult is it to walk the dog? *23rd European Workshop on Computational Geometry (EuroCG)*, pages 170–173, 2007. Graz, Austria.

[12] K. Buchin, M. Buchin, W. Meulemans, and W. Mulzer. Four Soviets walk the dog - with an application to Alt's conjecture. In *Proceedings of the 25th ACM-SIAM Symposium on Discrete Algorithms*, pages 1399–1413, 2014.

[13] D. Chen, A. Driemel, L. J. Guibas, A. Nguyen, and C. Wenk. Approximate map matching with respect to the Fréchet distance. In *Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 75–83. SIAM, 2011.

[14] R. Cole and U. Vishkin. The accelerated centroid decomposition technique for optimal parallel tree evaluation in logarithmic time. *Algorithmica*, 3:329–346, 1988.

[15] M. de Berg, A. F. Cook IV, and J. Gudmundsson. Fast Fréchet queries. *Computational Geometry – Theory and Applications*, 46(6):747–755, 2013.

[16] M. de Berg and M. Streppel. Approximate range searching using binary space partitions. *Computational Geometry – Theory and Applications*, 33:139–151, 2006.

[17] A. Driemel and S. Har-Peled. Jaywalking your dog: computing the Fréchet distance with shortcuts. *SIAM Journal on Computing*, 42:1830–1866, 2013.

[18] A. Driemel, S. Har-Peled, and C. Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete & Computational Geometry*, 48:94–127, 2012.

[19] M. Fréchet. Sur quelques points du calcul fonctionnel. *Rendiconti del Circolo Matematico di Palermo*, 22:1–74, 1906.

[20] J. Gudmundsson and M. Smid. Fréchet queries in geometric trees. In *Proceedings of the 21st European Symposium on Algorithms*, volume 8125 of *Lecture Notes in Computer Science*, pages 565–576, Berlin, 2013. Springer-Verlag.

[21] E. J. Keogh and M. J. Pazzani. Scaling up dynamic time warping to massive datasets. In *Proceedings of the 3rd European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 1–11. Springer, 1999.

[22] G. Narasimhan and M. Smid. *Geometric Spanner Networks*. Cambridge University Press, Cambridge, UK, 2007.

[23] C. A. Ratanamahatana and E. J. Keogh. Three myths about dynamic time warping data mining. In *Proceedings of the 5th SIAM International Data Mining Conference*. SIAM, 2005.

[24] E. Sriraghavendra, K. Karthik, and C. Bhattacharyya. Fréchet distance based approach for searching online handwritten documents. In *Proceedings of the 9th International Conference on Document Analysis and Recognition*, pages 461–465, 2007.

[25] C. Wenk. Shape matching in higher dimensions; chapter 5: Matching curves with respect to the Fréchet distance. *Dissertation, Freie Universität Berlin, Germany*, 2003.

[26] C. Wenk, R. Salas, and D. Pfoser. Addressing the need for map-matching speed: Localizing global curve-matching algorithms. In *Proceedings of the 18th Conference on Scientific and Statistical Database Management (SSDBM)*, pages 379–388, 2006.