Approximate Map Matching with respect to the Fréchet Distance

Daniel Chen * Anne Driemel † Leonidas J. Guibas * Andy Nguyen *
Carola Wenk ‡

Abstract

We extend recent results using curve simplification for approximating the Fréchet distance of realistic curves in near linear time to map matching: the problem of matching a curve in an embedded graph. We show that the theoretical bounds on the running time of the previous result still hold if only one of the curves is simplified during the course of the approximation algorithm. This enables our extension to the case of map matching under the assumption that the graph is ϕ-low density for a constant ϕ. We present experimental evidence for this assumption and implement the extended approximate matching algorithm. We show that it performs well on real world data, such as GPS traces and road networks of urban areas. In particular, it is able to perform matching tasks that took several hours with the exact matching algorithm in under a second.

1 Introduction

Given a polygonal curve $\pi : [0, 1] \to \mathbb{R}^d$ and an embedded graph G in $\mathbb{R}^d$ with edges embedded as straight line segments, we consider the problem of finding the path on G that is closest to π with respect to the Fréchet distance. This problem has become increasingly important over the past few years due to the proliferation of GPS tracking devices and applications. It has been studied before in [1, 3, 12] and both theoretical and experimental results have been obtained. Previous theoretical bounds on running time have been quadratic in the worst case, and hence are unsuitable for large, real world road maps.

In [3, 12] Wenk *et al.* implemented a practical weak Fréchet distance based map matching algorithm. Although for their examples the results are comparable to those obtained using the Fréchet distance, the weak Fréchet distance between two curves can still be arbitrarily smaller than the Fréchet distance. GPS traces can have self-intersecting loops, for instance at highway intersections, or follow a "zig-zag" shape, for instance when climbing a mountain. Those shapes would not be matched truthfully by the weak Fréchet distance. Motivated by a recent result for approximating the Fréchet distance between two polygonal curves in near linear time [7], we present a new algorithm for map matching that runs in near linear time for realistic input data. We also present a practical implementation of the algorithm with running times that are dramatically better than previous (strong) Fréchet distance based algorithms. The strength of our approach lies both in our theoretical guarantees and practical running times.

1.1 Input Model. We follow the approach of [5, 8, 10] in applying simple and realistic input models to place bounds on the running time of algorithms. In particular, we assume that the input curve π is c-packed, see [7], and the embedding of G is ϕ-low density, see [6]. We also implement an algorithm to calculate the low density constant ϕ and show experimental evidence that real world maps are ϕ-low density for a reasonable constant ϕ. For formal definitions of these input models, refer to Section 2.

1.2 Organization. In Section 2, we define the problem, revisit the definition of the input models low density and packedness, provide experimental justification for the input assumptions, and review foundational work. In Section 3, we explain the essential ideas of the map matching algorithm and provide a proof of theoretical running time bounds under the input assumptions. In Section 4, we summarize our experimental results and compare the performance of our algorithm to that of previous algorithms for the weak Fréchet distance.

*Computer Science Department, Stanford University, Palo Alto, CA 94305.
{danielc,guibas}@cs.stanford.edu, tanonev@stanford.edu

†Department of Information and Computing Sciences, Utrecht University, Netherlands, anne@cs.uu.nl. This work has been supported by the Netherlands Organisation for Scientific Research (NWO) under RIMGA.

‡Department of Computer Science, University of Texas at San Antonio, USA, carola@cs.utsa.edu. This work has been partially supported by the National Science Foundation grant NSF CAREER CCF-0643597.

2 Preliminaries

2.1 Input Model. The input to our algorithm is a polygonal curve π in $\mathbb{R}^d$ with p line segments and an embedded graph $G = (V, E)$ with q vertices equipped with a straight line embedding in $\mathbb{R}^d$. As in [1], the vertices $V = \{1, \ldots, q\}$ are embedded as points $\{v_1, \ldots, v_q\} \in \mathbb{R}^d$ and each edge $(i, j) \in E$ is embedded as a straight line segment $s_{i,j} = \overline{v_i v_j}$. For our theoretical bounds, we assume in addition that π is c-packed, as defined below:

Definition 2.1. (c-packed curve) *A curve π is c-packed if for any ball $\mathbf{b}(p, r)$, it holds that $\|\pi \cap \mathbf{b}(p, r)\| \leq cr$.*

This is a fairly general and realistic assumption for curves, as discussed in [7]. We also assume that G is ϕ-low density, as defined below:

Definition 2.2. (ϕ-low-density graph) *A graph G is* ϕ-low-density *if for any point p and any radius $r > 0$, the ball $\mathbf{b}(p, r)$ intersects at most ϕ edges of G that are longer than r.*

We experimentally verified that maps of real world cities are ϕ-low density for constant ϕ. The results are summarized in Figure 1. We computed a low density constant of 28 for a map of San Francisco. However, we remark that for most of the road network, the low density constant is much lower. The ball that contained 28 segments of length greater than its own radius happened to be a degenerate case in which many lanes were split up into separate edges, see Figure 2. Because of the way the low density input model is defined, these values do not grow with the size of the considered area within a city.

2.2 Computing the low density constant. To compute the values in Figure 1, we followed a simplified version of Algorithm 5.3 of [11].

Let E be the edges of the embedded graph that represents the map of the city. The algorithm iterates over the edges from longest to shortest. For each edge s_i, the algorithm computes the subset of edges $U_i \subseteq E$, that are longer than s_i and within distance $2\|s_i\|$ from s_i, where $\|s_i\|$ is the length of the edge s_i. Next, the algorithm computes the maximal number k_i of overlapping objects in the set of Minkowski sums

$$M_i = \{s \oplus \mathbf{b}_i \mid s \in U_i \vee s = s_i\},$$

where $\mathbf{b}_i$ is the ball of radius $\|s_i\|$ centered at the origin. The value of k_i is equal to the the maximum number of edges in E of length at least $\|s_i\|$ that can be covered by a disk of radius $\|s_i\|$.

Because the algorithm iterates over the edges from largest to smallest, the addition of any one edge cannot increase the current maximum value by more than one, i.e., $k_{i-1} \leq k_i \leq k_{i-1} + 1$. Therefore, to determine the value of k_i, it is sufficient to test if the $(k_{i-1} + 1)$-set of the current arrangement is non-empty. After iterating over all edges of E, we output that the set is k_n-low density.

The runtime of this algorithm is dominated by two factors: the time it takes to compute the U_i, which can be implemented as n range queries, and the time it takes to compute the $(k_{i-1} + 1)$-set of M_i for $1 \leq i \leq n$. However, a simple packing argument implies that $|M_i| \leq O(\phi)$ if E is ϕ-low density.

2.3 Problem Definition. First, we define the Fréchet distance:

Definition 2.3. (Fréchet distance) *For two curves given as continuous maps $\pi_1 : [0, p_1] \to \mathbb{R}^d$ and $\pi_2 : [0, p_2] \to \mathbb{R}^d$, the* Fréchet distance *is defined as*

$$\mathbf{d}_{\mathcal{F}}(\pi_1, \pi_2) := \inf_{\substack{\alpha:[0,1]\to[0,p_1] \\ \beta:[0,1]\to[0,p_2]}} \max_{t\in[0,1]} \|\pi_1(\alpha(t)) - \pi_2(\beta(t))\|_2$$

where α, β range over all continuous and monotonically increasing functions with $\alpha(0) = 0$, $\alpha(1) = p_1$, $\beta(0) = 0$ and $\beta(1) = p_2$.

We note that the Fréchet distance is symmetric and satisfies the triangle equality. A variation of the Fréchet distance known as the weak Fréchet distance drops the monotonicity requirements for α and β. We follow the convention that π is parameterized as $\pi : [0, p] \to \mathbb{R}^d$ and can be split into p line segments $\overline{\pi}_i = \pi|_{[i,i+1]}$. We assume that each line segment $s = \overline{AB}$, whether considered as an edge from the graph or as a part of the polygonal curve, is parameterized by its natural parameterization $s(\lambda) = (1 - \lambda)A + \lambda B$. The problem is then to find a path σ in G that minimizes the Fréchet distance $\mathbf{d}_{\mathcal{F}}(\pi, \sigma)$.

2.4 Free Space. The free space of two parameterized objects, first defined in [2], is a standard concept used in many Frechet distance computation algorithms. Intuitively, the free space is the sublevel set of a distance function on the parameter space of the objects. The Fréchet distance and its variants are generally computed by following certain paths through the free space.

More concretely, for two polygonal curves π_1 and π_2 the free space is defined to be $\mathcal{D}_{\leq\delta}(\pi_1, \pi_2) := \{(s, t) \in [0, p_1] \times [0, p_2] \mid \|\pi_1(s) - \pi_2(t)\| \leq \delta\}$. It can be verified that $\mathbf{d}_{\mathcal{F}}(\pi_1, \pi_2) \leq \delta$ if and only if there is a monotonic path in $\mathcal{D}_{\leq\delta}(\pi_1, \pi_2)$ from $(0, 0)$ to (p_1, p_2).

City	Low Density
San Francisco	28
Athens	17
Berlin	16
San Antonio	22

Figure 1: Low density constants of road networks of different cities.

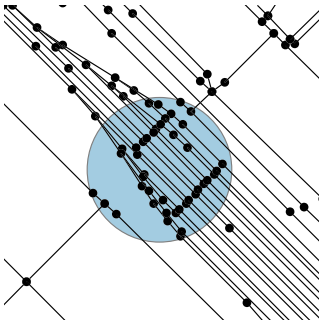

Figure 2: The ball in San Francisco defining the density constant.

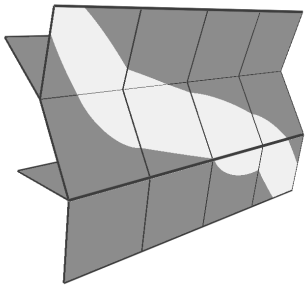

Figure 3: Free space between a curve and a graph.

2.5 Foundational Work. Our theoretical results are built upon that of [1] and [7]. Although these results include many details which are essential for proofs of correctness, we briefly summarize their basic approaches here:

In [1], Alt *et al.* extend the concept of the free space for the case of a path $\pi : [0, p] \to \mathbb{R}^d$, and an embedded graph G. The notation and description for the free space between a curve and a graph is as follows. First, we define

$$\mathcal{D}_j := \mathcal{D}_{\leq\delta}(\pi, v_j) = \{s \in [0, p] \mid \|\pi(s) - v_j\| \leq \delta\},$$

where v_j is a vertex of the graph G and

$$\mathcal{D}_{i,j} := \mathcal{D}_{\leq\delta}(\pi, s_{i,j}),$$

where $s_{i,j}$ is an edge of G. Then, the free space $\mathcal{D}_{\leq\delta}(\pi, G)$ of π and G can be viewed as the free space diagrams $\mathcal{D}_{i,j}$ glued together at the one dimensional free space diagrams $\mathcal{D}_j$. Alt *et al.* call this the *free space surface*. For an illustration, see Figure 3. Let $\mathbf{L_j}$ and $\mathbf{R_j}$ be the left endpoint and the right endpoint of $\mathcal{D}_j$. It can be verified that there exists a path σ in G where $\mathbf{d}_{\mathcal{F}}(\pi, \sigma) \leq \delta$ if and only if there exists a path ρ in $\mathcal{D}_{\leq\delta}(\pi, G)$ from some left corner $\mathbf{L_k}$ to some right corner $\mathbf{R_j}$ such that $\rho \setminus (\rho \cap (\bigcup_i \mathcal{D}_i))$ consists of monotonic path-connected components. For convenience, we call such paths *map monotonic*.

To decide whether a map monotonic path exists, [1] first computes *reachability pointers* (see Section 3.1.2), which encode portions of the path that can be matched to each edge in the graph. Then, a sweep algorithm is performed where the sweep increases with the parameter of the path and events are processed and discovered using the reachability pointers.

In [7], Driemel *et al.* introduce a framework for speeding up approximate Fréchet distance computations by using curve simplification and realistic input models to bound the complexity of the free space. In particular, their results yield a linear time algorithm for c-packed curves. Their approach begins by finding a linear sized set of approximate distances between the vertices of the curves using a well separated pairs decomposition. Then, a binary search over those values using the decision procedure, finds a suitable level of simplification for approximating the solution efficiently. In Section 3.3 we show that the bounds on the complexity of the free space still hold if only one of the curves is being simplified, which enables the extension to the map matching case.

3 Algorithm

3.1 Decision Procedure. We first describe an exact decision procedure based on the algorithm described in [1] that will serve as a building block for our complete algorithm.

3.1.1 Computing the Relevant Free Space. Let the *relevant free space* $\mathcal{R}_{\leq\delta}(\pi, G)$ be the set of points of $\mathcal{D}_{\leq\delta}(\pi, G)$ reachable from some $\mathbf{L_k}$ by some path, not necessarily map-monotonic. We note that to decide whether $\min_{\sigma \in G} \mathbf{d}_{\mathcal{F}}(\pi, \sigma) \leq \delta$, we only need to consider the relevant free space $\mathcal{R}_{\leq\delta}(\pi, G)$.

As in [1], we decompose two dimensional free spaces into cells corresponding to the free space between two segments. We also decompose one dimensional free spaces into intervals corresponding to the free space between a segment and a point. Now, let the number of cells with non-empty intersection with $\mathcal{R}_{\leq\delta}(A, B)$ be denoted $\mathcal{C}_{\leq\delta}(A, B)$. Then, we have the following easy lemma:

LEMMA 3.1. *All non-empty $\mathcal{R}_{\leq\delta}(\pi, v_j)$ and $\mathcal{R}_{\leq\delta}(\pi, s_{i,j})$ can be computed as ordered lists in time $O(q + \mathcal{C}_{\leq\delta}(\pi, G) \log p)$ and space $O(q + \mathcal{C}_{\leq\delta}(\pi, G))$.*

Proof. [Proof Sketch] First, we find the vertices v of G such that $\mathcal{R}_{\leq\delta}(\pi(0), v)$ is non-empty in $O(q)$ time. Then, we simulate depth first search on a graph where the vertices correspond to the cells $\mathcal{D}_{\leq\delta}(\overline{\pi(k)\pi(k+1)}, \overline{v_i v_j})$ and edges correspond to non-empty boundaries of these cells. To ensure $O(q + \mathcal{C}_{\leq\delta}(\pi, G))$ space, for each $\mathcal{D}_i$ we can maintain the corresponding edges in standard one-dimensional search trees ordered by their cell index along π. Hence, we have the additional factor of $\log p$. In practice, however, we used hashing by the cell index.

We show that we can decide whether there exists a map monotonic path in $\mathcal{R}_{\leq\delta}(\pi, G)$ from some $\mathbf{L_k}$ to some $\mathbf{R_j}$ in $O(q+\mathcal{C}_{\leq\delta}(\pi, G)\log(pq))$ time. We follow the approach of [1] in first computing reachability pointers, and then solving the problem using a sweep algorithm.

3.1.2 Computing Reachability Pointers. Let $\mathcal{R}_i$ and $\mathcal{R}_{i,j}$ be defined analogous to $\mathcal{D}_i$ and $\mathcal{D}_{i,j}$ above. For I a continuous interval of $\mathcal{R}_i$, let the reachability pointers $l_{i,j}(I)$ and $r_{i,j}(I)$ be the leftmost and rightmost points, respectively, of $\mathcal{R}_j$ that can be reached from some point in I by a monotonic path in $\mathcal{R}_{i,j}$. For an illustration, see Figure 4. Then, we have:

LEMMA 3.2. *Let $s_{i,j} \in E$ and let B_k be defined as $[k, k+1] \cap \mathcal{R}_i$. Assume that $\mathcal{R}_{\leq\delta}(\pi, v_i)$, $\mathcal{R}_{\leq\delta}(\pi, v_j)$ and $\mathcal{R}_{\leq\delta}(\pi, s_{i,j})$ have been computed as ordered lists. Then, all pointers $l_{i,j}(B_k)$ and $r_{i,j}(B_k)$ for $0 \leq k \leq p-1$ can be computed in $O(\mathcal{C}_{\leq\delta}(\pi, s_{i,j}))$ time.*

Proof. [Proof Sketch] Let

$$\mathcal{R}_{\leq\delta}(\pi, s_{i,j})_1, \mathcal{R}_{\leq\delta}(\pi, s_{i,j})_2, \ldots \mathcal{R}_{\leq\delta}(\pi, s_{i,j})_l$$

be the decomposition of $\mathcal{R}_{\leq\delta}(\pi, s_{i,j})$ into maximal sequences of consecutive cells and let

$$\mathcal{C}_{\leq\delta}(\pi, s_{i,j})_1, \mathcal{C}_{\leq\delta}(\pi, s_{i,j})_2, \ldots \mathcal{C}_{\leq\delta}(\pi, s_{i,j})_l$$

be their corresponding sizes.

By Lemma 3 of [1], the pointers $l_{i,j}(B_k)$ and $r_{i,j}(B_k)$ for $B_k \in \mathcal{R}_{\leq\delta}(\pi, s_{i,j})_m$ can be computed in $O(\mathcal{C}_{\leq\delta}(\pi, s_{i,j})_m)$ time. Thus, the pointers for all B_k can be computed in

$$\begin{aligned}\sum_{m=1}^{l} O(\mathcal{C}_{\leq\delta}(\pi, s_{i,j})_m) &= O\left(\sum_{m=1}^{l} \mathcal{C}_{\leq\delta}(\pi, s_{i,j})_m\right) \\ &= O(\mathcal{C}_{\leq\delta}(\pi, s_{i,j})).\end{aligned}$$

3.1.3 Sweep algorithm. The sweep algorithm is the same as that of [1], except we use the sparse lists created as in the previous subsection. We will have the following result:

THEOREM 3.1. *Let π be a polygonal curve of complexity p and G be a graph with q vertices and $|E|$ edges. Then, there is an algorithm* **decider**(π, G, δ) *that decides whether there is a path σ in G such that $\mathbf{d}_{\mathcal{F}}(\pi, \sigma) \leq \delta$ in $O(q + \mathcal{C}_{\leq\delta}(\pi, G)\log(pq))$ time and $O(q + \mathcal{C}_{\leq\delta}(\pi, G))$ space. If there is such a σ,* **decider** *also returns it.*

Proof. [Proof Sketch] Creating the reachability pointers takes $O(q + \mathcal{C}_{\leq\delta}(\pi, G)\log p)$ time. The sweep algorithm processes $O(\mathcal{C}_{\leq\delta}(\pi, G))$ events and each event takes $O(\log q)$ time to process. The space required is dominated by that of the reachability pointers which use $O(q + \mathcal{C}_{\leq\delta}(\pi, G))$ space.

3.2 Approximate Decision Procedure. We follow [7] in their framework of turning the decision procedure into a more efficient approximate decision procedure by first simplifying the curves. Our version differs in that we only simplify the curve π while we leave G intact. We first state the simple μ-simplification scheme from [7]:

DEFINITION 3.1. (μ-SIMPLIFICATION) *We denote with* $\text{simpl}(\pi, \mu)$ *the simplification of a polygonal curve π obtained by taking the first vertex of π, scanning through π and at each step only taking the next vertex that is in distance at least μ from the previously taken vertex.*

It is easy to see that each segment of $\text{simpl}(\pi, \mu)$ is of length at least μ and $\mathbf{d}_{\mathcal{F}}(\pi, \text{simpl}(\pi, \mu)) \leq \mu$. Then, the following "fuzzy" decision procedure can be obtained:

LEMMA 3.3. *Let π be a polygonal curve of complexity p and G be a graph with q vertices and $|E|$ edges. Let $\epsilon, \delta > 0$ be parameters. Then, we can construct an algorithm* **fuzzyDecider**$(\pi, G, \delta, \epsilon)$ *which runs in $O(q + \mathcal{C}_{\leq\delta}(\text{simpl}(\pi, \mu), G)\log(pq))$ time, where $\mu = (\epsilon/2)\delta$, and outputs one of the following:*

1. *If $\min_{\sigma\in G} \mathbf{d}_{\mathcal{F}}(\pi, \sigma) \leq \delta$, then the algorithm outputs reparameterizations to match π and $\sigma \in G$ within Fréchet distance $(1+\epsilon)\delta$.*
2. *If $\min_{\sigma\in G} \mathbf{d}_{\mathcal{F}}(\pi, \sigma) > (1+\epsilon)\delta$, then the algorithm outputs "$\min_{\sigma\in G} \mathbf{d}_{\mathcal{F}}(\pi, \sigma) > \delta$".*
3. *Otherwise, the algorithm outputs either of the above outcomes.*

Proof. [Proof Sketch] The proof is the same as that of [7], but we use $\mu = (\epsilon/2)\delta$ and $\delta' = \delta + \mu$ and appeal to Theorem 3.1 to decide whether $\min_{\sigma\in G} \mathbf{d}_{\mathcal{F}}(\text{simpl}(\pi, \mu), \sigma) \leq \delta'$.

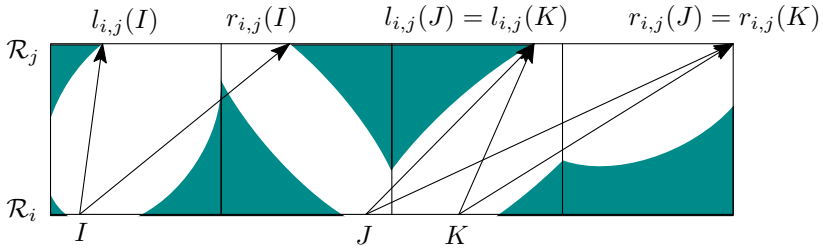

Figure 4: Reachability Pointers.

Just as in [7], we can use this approximate decider to guide a binary search in a precise way by calling it twice with different parameters. If the precise decision is not possible, then the decider returns a $(1+\epsilon)$ approximation directly.

LEMMA 3.4. *Let π be a polygonal curve of complexity p and G be a graph with q vertices and $|E|$ edges. Let $\epsilon, \delta > 0$ be parameters. Then, there is an algorithm* **approxDecider**$(\pi, G, \delta, \epsilon)$ *that in $O(q + \mathcal{C}_{\leq\delta}(\text{simpl}(\pi,\mu), G)\log(pq))$ time, where $\mu = O(\epsilon\delta)$, returns either:*

1. *A $(1+\epsilon)$-approximation to $\min_{\sigma\in G} \mathbf{d}_{\mathcal{F}}(\pi,\sigma)$*
2. *$\min_{\sigma\in G} \mathbf{d}_{\mathcal{F}}(\pi,\sigma) < \delta$.*
3. *$\min_{\sigma\in G} \mathbf{d}_{\mathcal{F}}(\pi,\sigma) > \delta$.*

Proof. [Proof Sketch] The proof is the same as that of [7], but we use the fuzzy decider procedure from Lemma 3.3.

3.3 Bounding the Complexity of the Relevant Free Space. First, we note the following lemma from [7]:

LEMMA 3.5. *Let π be a c-packed curve, $\mu > 0$, and $\pi' = \text{simpl}(\pi,\mu)$. Then, π' is a $6c$-packed curve.*

Then, we show that the complexity of the free space of the simplified curve with a graph $\mathcal{C}_{\leq\delta}(\text{simpl}(\pi,\mu), G)$ is linear in the complexity of π and G when π is a c-packed curve and G is a ϕ-low-density graph, for $\mu = \Omega(\epsilon\delta)$.

LEMMA 3.6. *Given a c-packed curve π with p edges and a graph G with $|E|$ edges in $\mathbb{R}^d$, it holds that*

$$\mathcal{C}_{\leq\delta}(\text{simpl}(\pi,\mu), G) \;=\; O(\phi p/\epsilon^d + c|E|/\epsilon),$$

for $\mu = \Omega(\epsilon\delta)$.

Proof. Let $\delta \geq 0$, $\mu = \epsilon\delta$ and $\pi' = \text{simpl}(\pi,\mu)$. $\mathcal{D}_{\leq\delta}(\pi', G)$ is composed of cells $\mathcal{D}_{\leq\delta}(\overline{\pi'}_i, s_{j,k})$ for $i = 1, 2, \ldots, p$ and $(i,j) \in E$ and thus $\mathcal{C}_{\leq\delta}(\pi', G)$ is upper bounded by the number of non-empty cells. A cell $\mathcal{D}_{\leq\delta}(\overline{\pi'}_i, s_{j,k})$ is non-empty if and only if there are points on $\overline{\pi'}_i$ and $s_{j,k}$ within δ of each other. We charge each such non-empty cell to the shorter of the two segments. We divide the analysis into three cases:

- Case 1: Some $\overline{\pi'}_i$ is charged. In this case, we consider the ball of radius $\|\overline{\pi'}_i\|/2 + \delta < \|\overline{\pi'}_i\|/2 + \|\overline{\pi'}_i\|/\epsilon = (1/2 + 1/\epsilon)\|\overline{\pi'}_i\|$ around the midpoint of $\overline{\pi'}$. Any $s_{j,k}$ that charges $\overline{\pi'}_i$ must lie in this ball and be of length at least $\|\overline{\pi'}_i\|$. However, we note that this ball can be covered by $O(1/\epsilon^d)$ balls of radius $\|\overline{\pi'}_i\|$ and hence there are at most $O(\phi/\epsilon^d)$ such $s_{j,k}$. Hence, the total number of charges due to Case 1 is $O(\phi p/\epsilon^d)$.
- Case 2: Some $s_{j,k}$ is charged and $\|s_{j,k}\| \geq \mu$. Here we use the same ball of radius $\frac{3}{2}\|s_{j,k}\| + \delta$ around the midpoint of $s_{j,k}$. Then, using the same argument as in [7], the number of charges to $s_{j,k}$ is at most $9c + \frac{6c\delta}{\mu} = O(c/\epsilon)$.
- Case 3: Some $s_{j,k}$ is charged and $\|s_{j,k}\| < \mu$. Here we use the ball $\mathbf{b}$ of radius $\frac{1}{2}\|s_{j,k}\| + \mu + \delta$ around the midpoint of $s_{j,k}$. Then every segment of π' is of length at least μ and furthermore the intersection of the segment with the ball is of length at least μ. Therefore, the number of charges to $s_{j,k}$ is $\frac{\|\pi' \cap \mathbf{b}\|}{\mu} \leq \frac{6c(\frac{1}{2}\|s_{j,k}\| + \mu + \delta)}{\mu} \leq 9c + \frac{6c\delta}{\mu} = O(c/\epsilon)$.

The total number of charges in cases 2 and 3 are $O(c|E|/\epsilon)$.

COROLLARY 3.1. *Given a c-packed curve π with p edges and a graph G with $|E|$ edges in $\mathbb{R}^d$, the algorithm* **approxDecider**$(\pi, G, \delta, \epsilon)$ *and the algorithm* **fuzzyDecider**$(\pi, G, \delta, \epsilon)$ *take time in*

$$O(q + (\phi p/\epsilon^d + c|E|/\epsilon)\log(pq)).$$

3.4 $(1+\epsilon)$-Approximation Algorithm. Critical values are the values of δ or μ, for which the structure of $\mathcal{D}_{\leq\delta}(\text{simpl}(\pi,\mu),G)$ might change to allow or disallow a map monotonic path. They can be used to guide the search for the Fréchet distance, since the optimal value has to coincide with one of them. There are three types of critical values: (1) Vertex-edge critical values, (2) Monotonicity critical values, and (3) Simplification critical values. Vertex-edge critical values are simply the distances between vertices and edges, monotonicity critical values are the values of δ for which a map monotonic path opens up in the free space, and simplification critical values are the values of μ where $\text{simpl}(\pi,\mu)$ changes. We note that the simplification critical values are a subset of the pairwise distances of all points. For a bound on monotonicity critical values, [7] shows the following important lemma:

LEMMA 3.7. *Let x be the monotonicity critical value. Then, there exists a number y such that $y/2 \leq x \leq 3y$ and y is either a distance between vertices or a vertex-edge critical value.*

We also note the approximate distance selection algorithm used in [7]:

LEMMA 3.8. *Given a set P of n points, let D be the set of all pairwise distances. Then, there is an algorithm* **approxDistances**(P) *that can compute in $O(n\log n)$ time a set Z of $O(n)$ numbers such that for any $y \in D$, there exists numbers $x, x' \in Z$ such that $x \leq y \leq x' \leq 2x$.*

Our resulting algorithm is then described as follows:

ALGORITHM 3.1. **Require:** Polygonal curve π, embedded graph $G=(V,E)$ with straight line segment edges, $\epsilon > 0$.
Ensure: An $(1+\epsilon)$ approximation to $\min_{\sigma\in G}\mathbf{d}_{\mathcal{F}}(\pi,\sigma)$ is returned.
1: Let $Z =$ **approxDistances**$(V(\pi)\cup V(G))$.
2: Use **approxDecider**$(\pi,G,\cdot,1)$ to perform binary search on Z to either return a 2 approximation or find an interval $[\alpha,\beta]$ where $\min_{\sigma\in G}\mathbf{d}_{\mathcal{F}}(\pi,\sigma)$ lies. If it returns a 2 approximation γ, let $[\alpha,\beta]=[\gamma/2,\gamma]$.
3: **if** $\beta > 2\alpha$ **then**
4: Let Y be the vertex-edge critical values for $\text{simpl}(\pi,\alpha)$ and G less than or equal to β.
5: Use **decider**$(\text{simpl}(\pi,\alpha),G,\cdot)$ to perform binary search on Y to find an interval $[\theta,\iota]$ that does not contain any simplification critical value or vertex-edge critical value.
6: **if decider**$(\text{simpl}(\pi,\alpha),G,3\theta)$ **then**
7: $\beta = 3\theta+\alpha$.
8: **else**
9: $\beta = \iota+\alpha$.
10: **end if**
11: $\alpha=\beta/7$. $\{\beta$ is a 7-approximation.$\}$
12: **end if**
13: **while** $\beta > (1+\epsilon)\alpha$ **do**
14: $\Delta=(\beta-\alpha)/4$.
15: $x_i=\alpha+i\Delta$ for $i=1,2,3$.
16: $\epsilon'=\Delta/(4\alpha)$. $\{1/\epsilon'$ at least doubles in each iteration.$\}$
17: **if fuzzyDecider**(π,G,x_1,ϵ') returns $\min_{\sigma\in G}\mathbf{d}_{\mathcal{F}}(\pi,\sigma)\leq(1+\epsilon')x_1$ **then**
18: $\beta = x_2$.
19: **else if fuzzyDecider**(π,G,x_2,ϵ') returns $\min_{\sigma\in G}\mathbf{d}_{\mathcal{F}}(\pi,\sigma)\leq(1+\epsilon')x_2$ **then**
20: $[\alpha,\beta]=[x_1,x_3]$.
21: **else**
22: $\alpha = x_2$.
23: **end if**
24: **end while**
25: Return β.

By Lemma 3.8, Line 1 takes time $O((p+q)\log(p+q))$. Since Z is of size $O(p+q)$, Line 2 takes $O(q+(\phi p+c|E|)\log(pq)\log(p+q))$ time, by Corollary 3.1. If Line 3 is true, then there is no simplification critical value in (α,β), because otherwise Lemma 3.8 would be contradicted. Therefore, $\text{simpl}(\pi,\alpha) = \text{simpl}(\pi,\beta)$, and, by Lemma 3.6, Line 4 takes $O(q+\mathcal{C}_{\leq\beta}(\text{simpl}(\pi,\beta),G)\log(pq)) = O(q+(\phi p+c|E|)\log(pq))$ time and Lines 5 and 6 have the same running time as Line 2. Line 5 finds an interval $[\theta,\iota]$ that does not contain any simplification critical value or any vertex-edge critical value. By Lemma 3.7, all monotonicity critical points in $[\theta,\iota]$ are either in $[\theta,3\theta]$ or $[\iota/2,\iota]$ and therefore either 3θ or ι is a 3-approximation to $\min_{\sigma\in G}\mathbf{d}_{\mathcal{F}}(\text{simpl}(\pi,\alpha),\sigma)$. In Line 7 and 9 we add α to ensure that $\beta > \min_{\sigma\in G}\mathbf{d}_{\mathcal{F}}(\pi,\sigma)$. We argue that

$$\begin{aligned}\beta &\leq \alpha+3\min_{\sigma\in G}\mathbf{d}_{\mathcal{F}}(\text{simpl}(\pi,\alpha),\sigma)\\ &\leq \alpha+3(\min_{\sigma\in G}\mathbf{d}_{\mathcal{F}}(\pi,\sigma)+\alpha)\\ &\leq 7\min_{\sigma\in G}\mathbf{d}_{\mathcal{F}}(\pi,\sigma)\end{aligned}$$

since the Fréchet distance of a curve and its μ-simplification is at most μ and we know that $\alpha \leq \min_{\sigma\in G}\mathbf{d}_{\mathcal{F}}(\pi,\sigma)$. Therefore, by the end of Line 11, we have an interval $[\alpha,\beta]$ that contains $\min_{\sigma\in G}\mathbf{d}_{\mathcal{F}}(\pi,\sigma)$ and such that β is a 7-approximation of $\min_{\sigma\in G}\mathbf{d}_{\mathcal{F}}(\pi,\sigma)$. Therefore, in the while loop, $\beta-\alpha$ is initially at most 6α and hence after at most $M=\lceil\log(6/\epsilon)\rceil$ iterations the while loop will terminate. It is not difficult to see that we end with β being a $(1+\epsilon)$-approximation.

To bound the running time, we observe that it is

$$O\left(\sum_{i=1}^{M}(q+\mathcal{C}_{\leq\beta_i}(\text{simpl}(\pi,\epsilon'_i\beta_i),G)\log(pq))\right)$$

where ϵ'_i is the value of ϵ' in the ith iteration. By Lemma 3.6,

$$\mathcal{C}_{\leq\beta_i}(\text{simpl}(\pi,\epsilon'_i\beta_i),G) \;\leq\; O(\phi p/\epsilon'^d_i + c|E|/\epsilon'_i),$$

and since $1/\epsilon'_i$ increases exponentially with i, and the summation term is at least linear in $1/\epsilon'_i$,

$$q+(\phi p/\epsilon'^d_j + c|E|/\epsilon'_j)\log(pq)$$

dominates this sum where j is the last iteration. It is easy to check that $\epsilon'_j > c_1\epsilon$ for a constant c_1, and therefore, we can summarize the runtime in the following theorem:

THEOREM 3.2. *Given a c-packed polygonal curve of complexity p and a ϕ-low density graph G with q vertices and $|E|$ edges in $\mathbb{R}^d$, and a parameter $\epsilon > 0$, one can $(1+\epsilon)$-approximate the $\min_{\sigma\in G}\mathbf{d}_{\mathcal{F}}(\pi,\sigma)$ in $O((p+q)\log(p+q)+(\phi p+c|E|)\log(pq)\log(p+q)+(\phi p/\epsilon^d + c|E|/\epsilon)\log(pq))$ time.*

4 Implementation and Experiments

4.1 Implementation. We implemented our algorithm in C++ using only the C++ Standard Library and hence our code is portable and compiles on multiple platforms. Our initial implementation followed Algorithm 3.1 faithfully. We implemented **approxDistances** using a well separated pairs decomposition. The implementation followed [4]'s original fair split tree based approach. However, preliminary experiments showed that 95% of the computation time was spent in the approximate distance computations, while for real world data sets of bounded spread, it is not a difficult problem to guess these values. Therefore, we replaced Lines 1 to 11 with a heuristic to guess a 2-approximation. Since our data is from real world data sets, the GPS traces never map directly on the maps, and hence the distances from the vertices of the traces to the vertices of the map are positive. Therefore, we first let β be the distance from the first vertex of the trace to the closest vertex of the map. Then, we doubled β until **approxDecider**(π,G,β,ϵ) returned $\min_{\sigma\in G}\mathbf{d}_{\mathcal{F}}(\pi,\sigma) < \beta$. In this fashion, we obtained a 2-approximation and the rest of the algorithm was implemented as written.

4.2 Results. Previous implementations of (strong) Fréchet map matching required several hours to match one trajectory to a large real world map [12]. Therefore, we were only able to compare the performance of our algorithm with previous Java implementations of weak Fréchet map matching. The testing platform for our algorithm was a Dell Precision 380 with a 3.2 GHz Pentium D and 2GB of RAM running Linux and the testing platform for the weak Fréchet matching was the same computer but running Windows. In general, differences in performance may be due to the implementation language and platform, but without heavy paging and with our specific problem, it is reasonable to assume that running times differing in several orders of magnitude provide evidence of differences in how the algorithms themselves perform.

We performed experiments on two data sets, one from Berlin, Germany, and the other from Athens, Greece. The data is a subset of the data collected and used in the study by Pfoser et al. [9]. The Berlin road map had 429,256 vertices and 818,632 edges and the Athens road map had 498,489 vertices and 1,000,040 edges. The GPS traces were collected in 2007 and consists of 955 traces from taxi fleets in Berlin and 12 traces from delivery trucks in Athens. The typical sampling rate is 30s, which is determined by the communication frequency of the fleet management system.

The table shows the results from our comparison. The implementation of the weak Fréchet distance was a version designed to be fast in practice, and hence included parameters that allowed the algorithm to cut the input curve into shorter pieces for various reasons. For example, when the matched Fréchet distance gets too large, the algorithm assumes that there is an outlier so some vertices are pruned. This parameter is called Dist Threshold in the table. Furthermore, if the time distance between two consecutive vertices gets too large, the algorithm cuts the curve into shorter pieces because it does not make sense to assume that the curve between the vertices should be comparable to a line segment. This parameter is called Time Diff in the table.

In general, the algorithms performed much slower on the Athens dataset, as the traces are much more noisy and the road map used did not account for a construction boom in Athens, and hence many roads are not contained in the map. The inaccuracy of the data resulted in the algorithms considering a much larger portion of the maps than otherwise necessary. We can see the effect of this in the number of pieces the Weak Fréchet implementation split the input curve into. Our algorithm does not have such optimizations implemented, but still manages to be comparable in performance to the weak Fréchet algorithm. When the weak Fréchet algorithm has the curve cutting turned off

Algorithm	Parameters	Dataset	Pieces	Runtime (ms)	
Weak	Time Diff = 90	Berlin	4.08	19332	avg
Fréchet			1	15	min
			55	652047	max
Weak	Dist Thresh = 150	Berlin	4.15	323	avg
Fréchet	Time Diff = 90		1	15	min
(optimized)			55	9672	max
Approx	Epsilon = 0.1	Berlin	N/A	348	avg
Fréchet			N/A	100	min
			N/A	2600	max
Weak	Time Diff = 90	Athens	20.92	1369371	avg
Fréchet			9	280610	min
			34	3946218	max
Weak	Dist Thresh = 150	Athens	22.5	30371	avg
Fréchet	Time Diff = 90		10	3875	min
(optimized)			37	112797	max
Approx	Epsilon = 0.1	Athens	N/A	32353	avg
Fréchet			N/A	780	min
			N/A	272410	max

Table 1: Comparison with weak Fréchet map matching algorithms from [12]. We were unable to have a comparison with the previous algorithm for (strong) Fréchet map matching because it was not able to match traces in a reasonable amount of time.

for distance threshold, our new algorithm is much faster. We hypothesize that our running times would improve even more if such optimizations were implemented.

5 Conclusion and Future Work

Our work provides evidence that the simplification approach of [7] can result in dramatically faster running times in practice. Map matching in practice often has much larger input data than that of simply comparing curves, and hence the performance of algorithms becomes much more important. Our extension of their algorithm to map matching not only resolves one of the open problems posed in the paper, but also provides a task for which their simplification approach can turn a previously impractical problem to something that can be solved in less than a second.

Our current implementation also does not have any specific optimizations to deal with outliers and inconsistent data. For future work, we plan to add such optimizations, along with spatial searching, and test how the result would perform in practice. We also see the opportunity to investigate simplification for weak Fréchet map matching, and to use our algorithm to consider proximity queries such as nearest neighbor searching with respect to the Fréchet distance. Another interesting direction for future work is to investigate whether it is possible to preprocess a map for sub-linear time map matching queries.

Acknowledgments

We would like to thank Sariel Har-Peled for helpful discussions.

References

[1] H. Alt, A. Efrat, G. Rote, and C. Wenk, *Matching planar maps*, J. Algorithms, 49 (2003), pp. 262–283.

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

[3] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk, *On Map-Matching Vehicle Tracking Data*, in Proc. 31st VLDB Conference, 2005, pp. 853–864.

[4] P. B. Callahan and S. R. Kosaraju, *A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields*, J. Assoc. Comput. Mach., 42 (1995), pp. 67–90.

[5] M. de Berg, *Linear size binary space partitions for uncluttered scenes*, Algorithmica, 28 (2000), pp. 353–366.

[6] M. de Berg, M. J. Katz, A F. van der Stappen, and J. Vleugels, *Realistic Input Models for Geometric Algorithms*, Algorithmica, 34 (2002), pp. 81–97.

[7] A. Driemel, S. Har-Peled, and C. Wenk, *Approximating the Fréchet Distance for Realistic Curves in Near Linear Time*, in Proc. 26th Annu. ACM Sympos. Comput. Geom., 2010.

[8] M. H. Overmars and A. F. van der Stappen, *Range searching and point location among fat objects*, J. Algorithms, 21:3 (1996), pp. 629–656.

[9] D. Pfoser, S. Brakatsoulas, P. Brosch, M. Umlauft, N. Tryfona, and G. Tsironis, *Dynamic travel time provision for road networks*, in Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems (ACM GIS), 2008.

[10] O. Schwarzkopf and J. Vleugels, *Range searching in low-density environments*, Inf. Process. Lett., 60:3 (1996), pp. 121–127.

[11] J. Vleugels, *On Fatness and Fitness - Realistic Input Models for Geometric Algorithms*, PhD thesis, Dept. Comput. Sci., Utrecht Univ., Utrecht, the Netherlands, 1997.

[12] C. Wenk, R. Salas, and D. Pfoser, *Addressing the Need for Map-Matching Speed: Localizing Global Curve-Matching Algorithms*, in Proc. 18th Int. Conf. Sci. Statis. Database Manag., 2006, pp. 879–888.