Fast Map Matching with Vertex-Monotone Fréchet Distance

Daniel Chen
Apple, Cupertino, CA, USA

Christian Sommer
Apple, Cupertino, CA, USA

Daniel Wolleb
Apple, Cupertino, CA, USA

Abstract

We study a generalization for map matching algorithms that includes both geometric approaches such as the Fréchet distance and global weight approaches such as those typically used by Hidden Markov Models. Through this perspective, we discovered an efficient map matching algorithm with respect to the *vertex-monotone Fréchet distance* while using a heuristic tie-breaker inspired by global weight methods. While the classical Fréchet distance requires parameterizations to be monotone, the vertex-monotone Fréchet distance allows backtracking within edges. Our analysis and experimental evaluations show that relaxing the monotonicity constraint enables significantly faster algorithms without significantly altering the resulting map matched paths.

2012 ACM Subject Classification Theory of computation → Shortest paths

Keywords and phrases Fréchet distance, map matching, minimum bottleneck path

Digital Object Identifier 10.4230/OASIcs.ATMOS.2021.10

1 Introduction

With the widespread availability of receivers for the Global Positioning System (GPS) in modern cars and smartphones came the rise of large databases of GPS traces (also called trajectories). These trajectory databases are valuable sources for various applications like map construction [8, 2, 25], map refinement and correction [42, 26], traffic estimation [6, 9, 35], travel time estimation [28, 47], dynamic routing [43, 14, 12, 13], ride sharing [18], mobility studies [19, 36], location mining [53], and many more. As a first step, GPS traces often get mapped to a road network via a non-trivial process called *map matching*. Beyond the applications mentioned above, map matching can also help with indexing the trajectory database to support fast retrieval of traces [30, 21].

Given a GPS trace as a sequence of (latitude, longitude) pairs, possibly equipped with time stamps and auxiliary information, a map matching algorithm is expected to return a connected sequence of road segments or edges (i.e., a path) in the road network that the input trace originally traversed. In some situations, the match is rather obvious, e.g., for a straight-line trace along an isolated stretch of road. The main challenge of map matching lies in the interplay of noisy GPS observations and dense road networks, especially in urban settings with complex intersections, highway crossings, and stacked roads, where skyscrapers may further block or otherwise interfere with the satellite signal. In such scenarios there can be many candidate paths for a given GPS trace, with several viable options for the path the device was actually traveling on.

Beyond qualitative aspects, another important consideration is the performance of a map matching algorithm. Contemporary trajectory databases are large, as is their growth, and many of the applications mentioned above typically benefit from low-latency processing of the most recent traces. Furthermore, road networks change over time, at least locally,

which in turn may require a decent amount of reprocessing of historical traces. Faster map matching performance can then be leveraged to achieve better tradeoffs between the number of cores required for map matching, the latency of the final result, and the amount of data that can be processed.

1.1 Related Work

Within the field of trajectory mining, map matching has been studied in many different models. For a general survey, we refer to Zheng [52]. For a survey on online map matching (where we are given only a few recent GPS points), we refer to Quddus et al. [39].

In this paper, we are interested in *offline* map matching, where the input consists of a typically longer GPS trace that should be matched to the map. Offline map matching allows to match many traces to a map simultaneously [31]. However, most papers, including our work, study the scenario where each trace is matched separately and independently.

A survey by Wei et al. [48, 49] distinguishes between incremental and global approaches. While incremental algorithms were successful at the SIGSPATIAL Cup 2012 [3, 45], global approaches are generally more accurate [7]. Weight-based, global approaches, like the Hidden Markov Model by Newson and Krumm [32], are widely used [24, 34] and also work in the online setting [23]. However, as Wei et al. [48, 49] argue, they require careful parameter tuning and make strong independence assumptions on the distribution of the GPS errors. Geometric approaches largely avoid parameter tuning by simply minimizing a distance function between the trace and a matching path.

The Fréchet distance is frequently used in many different variations for map matching [50, 11, 44, 4, 10] and other applications [27] and is also the main ingredient in our algorithm. Indeed, our algorithm borrows from much of the prior research into Fréchet distance based map matching, but our focus is on map matching high sampling rate and relatively low error GPS traces in a small number of microseconds per crumb, while quickly rejecting those with high error. There are many large data sets consistent with these properties, including the one in SIGSPATIAL Cup 2012, and they are often a more reliable source of information than high error or sparsely sampled GPS traces. By focusing on these data conditions and by relaxing the Fréchet distance, we are able to avoid more complex machinery in [50, 11, 44] and optimize absolute runtimes for low error traces.

There is also prior work on map matching with respect to the Fréchet distance while restricting candidates to shortest paths on the graph [10]. While our algorithm does not enforce such a restriction, we do prefer shorter routes when breaking ties. Indeed, map matching algorithms based purely on the Fréchet distance may sometimes choose an arbitrary matched path out of the set of paths with the same Fréchet distance. Wei et al. [48, 49] presented an algorithm that combines a Fréchet distance and a weight-based approach that prefers shorter and closer routes, and we build on their result with the goal of improving the running time without sacrificing accuracy.

1.2 Contributions

We present a novel map matching algorithm based on the vertex-monotone Fréchet distance [46, 27]. There are several ingredients that contribute substantially to performance and quality. We discuss the following three in more detail:

- Wenk et al. [50] compute the weak Fréchet distance by pruning the search space using the road network geometry. Pursuing an analogous approach, we get a significant performance boost.

- We present a novel trace simplification technique to further improve the running time.
- To distinguish between different paths of the same distance, we introduce a global weight function, similar to the one used by Wei et al. [48, 49], to achieve a high matching accuracy.

Finally, we propose a general framework that provides a unified view on both Hidden Markov Models and Fréchet distance approaches. We argue that they both can be seen as path searches in parametrization spaces, which motivates some of the tradeoffs we chose when designing our algorithm. In our experimental evaluation, we observe that our algorithm is significantly faster than previously published results, which makes it a viable and competitive alternative to popular HMM-based methods.

2 Preliminaries

2.1 Fréchet distances between curves

In map matching, we want to find a matching path through the road network that is closest to a given trace of a GPS device under some measure of similarity. To do this, we model both paths on the network and GPS traces as curves in $\mathbb{R}^2$. One natural distance function between curves in $\mathbb{R}^2$ is the *Fréchet distance*.

A illustrative mental picture for the Fréchet distance is the following: A dog and its owner go for a walk. The dog strolls along the first curve while the owner walks along the second curve going sometimes faster, sometimes slower, in an effort to keep the leash of the dog as short as possible for the entire path. The shortest possible leash length that allows both of them to traverse their paths while never walking backwards is precisely what is called the Fréchet distance between the two curves.

In the following, we define the Fréchet distance as well as three known variants and their relations (weak, discrete, and vertex-monotone Fréchet distance).

▶ **Definition 1** (Fréchet distance [20, 5]). *For two curves given as continuous maps* $\pi : [1, n] \to \mathbb{R}^2$ *and* $\sigma : [1, m] \to \mathbb{R}^2$*, the* Fréchet distance *is defined as*

$$d_{FD}(\pi, \sigma) = \inf_{\substack{f:[0,1]\to[1,n] \\ g:[0,1]\to[1,m]}} \max_{t\in[0,1]} \|\pi(f(t)) - \sigma(g(t))\|_2 , \tag{1}$$

where f *and* g *are continuous and monotonically increasing functions with* $f(0) = 1$*,* $f(1) = n$*,* $g(0) = 1$*,* $g(1) = m$*.*

If we omit the monotonicity constraint, i.e. allow the owner and the dog to backtrack along their curves, we get what is known as the *weak Fréchet distance*. While it can lead to efficient algorithms, as proposed by Wenk et al. [50], the weak Fréchet distance can be arbitrarily smaller than the Fréchet distance, as e.g. noted by Chen et al. [11]. In particular, a matched trace might correspond to driving back and forth on a one-way street to better match a loop in the trace trajectory, e.g. at a highway intersection.

Another way to relax the problem while still enforcing monotonicity is to look only at the vertex positions, assuming that both curves are polygonal. This means that the dog and the owner progress in discrete steps and in each step they each either jump to the next vertex or stay put. The main issue with the *discrete Fréchet distance* is that it can be large even for two curves that visually are very close if one of the two curves is sampled very coarsely. In particular, if a long, straight section of a highway is modeled as a single segment, even a trace that follows it very closely would have a large discrete Fréchet distance. One

way to handle this would be to supersample the geometry to a sufficiently high granularity, but this would increase the complexity of the free space diagram. Indeed, as we will see later on, reducing the complexity of the free space diagram through geometry simplification significantly improves our runtimes.

Finally, we introduce yet another variant of the Fréchet distance, which we use in our map matching algorithm. Consider the setting where the dog and the owner have to walk continuously on two polygonal curves, but where the monotonicity constraint is relaxed to allow for backtracking within each straight-line segment, but not past any vertex. This is known as the *vertex-monotone Fréchet distance*, which was defined by van Leusden [46]. Below is a formal definition (adding the boundary constraints missing in [46]).

▶ **Definition 2** (Vertex-monotone Fréchet distance [46]). *For two polygonal curves given as linearly-interpolated, continuous maps $\pi : [1, n] \to \mathbb{R}^2$ and $\sigma : [1, m] \to \mathbb{R}^2$, the* vertex-monotone Fréchet distance *is defined as*

$$d_{VMFD}(\pi, \sigma) = \inf_{\substack{f:[0,1]\to[1,n] \\ g:[0,1]\to[1,m]}} \max_{t\in[0,1]} \|\pi(f(t)) - \sigma(g(t))\|_2 , \tag{2}$$

where f and g are continuous with $f(0) = 1$, $f(1) = n$ and if $f(t) > i$ for any $t \in [0, 1]$ and $i \in [\![1, n]\!]$, then also $f(t') > i$ for any $t' > t$, and likewise for g.

We can observe the following order between all these distance functions for any pair of polygonal curves (here d_{WFD} refers to the weak Fréchet distance, and d_{DFD} refers to the discrete Fréchet distance):

$$d_{\mathrm{WFD}} \leq d_{\mathrm{VMFD}} \leq d_{\mathrm{FD}} \leq d_{\mathrm{DFD}} \tag{3}$$

This follows from the fact that the mapping functions f and g get more and more constrained from left to right. Moreover, we note that using the triangle inequality, we can also bound

$$d_{\mathrm{FD}} \leq d_{\mathrm{VMFD}} + D, \tag{4}$$

where D is the length of the longest line segment on π and σ.

Furthermore, note that we can bring d_{VMFD} arbitrarily close to d_{FD} by simply subdividing long segments of π and σ. Also note that d_{FD} can differ from d_{VMFD} even on undirected graphs because d_{FD} allows the mapping to switch directions only at the vertices.

2.2 Map matching problem

The map matching problem has two inputs: The first one is a *GPS trace* T that describes the trip of a driver as n points (also called *crumbs*), each specifying a position $p_i \in \mathbb{R}^2$ and monotonically increasing time stamps $t_i \in \mathbb{R}$. For simplicity, we assume that the times are normalized to $t_1 = 1$ and $t_n = n$. We model the route of the driver as a 2D-curve consisting of the polyline given by $p_1, \ldots, p_n$ and the time parametrization implied by $t_1, \ldots, t_n$, where for any time t with $t_i < t < t_{i+1}$, we have that $T(t)$ is the linear interpolation between p_i and p_{i+1}, see Figure 1.

The second input is a *road network*, given as a directed graph $G = (V, A)$ with vertices V and arcs A, where each vertex $v_i \in V$ has a location $\ell_i \in \mathbb{R}^2$, and each arc $a = (v_i, v_j) \in A$ has the *shape* of the straight line $\overline{\ell_i, \ell_j}$. Note that others often allow for arbitrary polylines as the shape of an arc, which we can simply model as subdividing arcs with vertices of degree two.

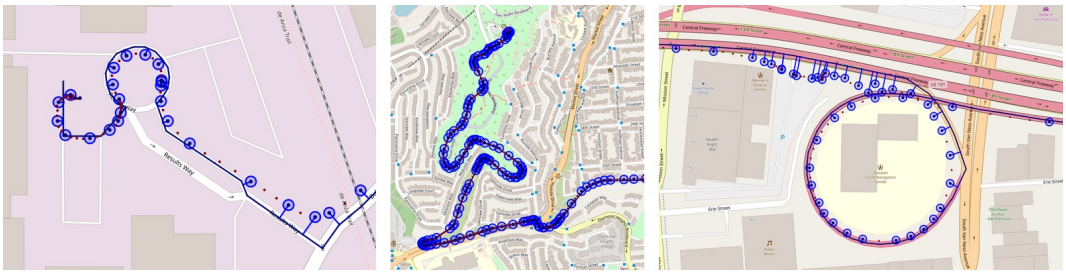

Figure 1 Visualization of three map matching situations. Crumbs shown in red were dropped in the simplification step prior to map matching. Crumbs shown in blue got matched to the road network. The dark blue path shows the matched path P, and the light blue lines show the matching of crumbs onto the road network. Sometimes, GPS is very accurate and the trace lines up perfectly with the geometry of the road network (middle). Oftentimes, the trace and the network do not align perfectly due to the distance to the middle of the road on the map (left) or due to noisy GPS measurements (right).

Let Π be the set of paths through G, where a path P is a sequence of m vertices $v_1, \ldots, v_m$ that are connected by arcs, i.e., $(v_i, v_{i+1}) \in A$ for $1 \leq i < m$. We also view P as the polygonal $2D$-curve described by the sequence of arcs and parametrized such that $P(i) = v_i$.

We are looking for a path $P \in \Pi$ that *best* reflects the journey that the driver took through the road network. To define what *best* means in this context, we look at two parametrizations: $f : [0, 1] \mapsto [1, n]$ for the trace T and $g : [0, 1] \mapsto [1, m]$ for the map matched path P. We now use the distance measures defined in Section 2.1 to restrict f and g, and to specify which path P we desire.

▶ **Definition 3** (Vertex-monotone Fréchet distance map matching). *Given trace T and network G, the* vertex-monotone Fréchet distance map matching problem *asks to find the path $P \in \Pi$ as well as parametrizations f and g such that $d_{VMFD}(T, P)$ is minimized.*

Note that for map matching, it is reasonable to relax the boundary condition of g in d_{VMFD} to $g(0) \in [1, 2]$ and $g(1) \in [m - 1, m]$ so as to allow for the beginning and end of the trace to be each matched to the interior of an arc.

2.3 Free-space diagram

An important tool to compute the Fréchet distance between curves and also for map matching is the so-called *free-space diagram* (also sometimes called *free-space surface*). It is defined as the sublevel set of the Euclidean distance function with respect to the parameter space of the trace and the graph. For a threshold $\epsilon > 0$, the free space $F_{\leq \epsilon}$ is the set of pairs of points, one on the trace T, one on the graph G, so that they are within distance ϵ of each other:

$$F_{\leq \epsilon}(T, G) = \{(p, q) \mid p \in T, q \in G, \|p - q\|_2 \leq \epsilon\}. \tag{5}$$

This notion is useful because searching for a map matching can now be seen as finding a path with certain properties through this free-space diagram. Namely, we are looking for a vertex-monotone path from (p_1, u) to (p_n, v) for some arcs $u, v \in A$. Let us call the individual elements of this Cartesian product *cells* and its borders *intervals* of the free-space diagram. We refer to Figure 2 for an illustration.

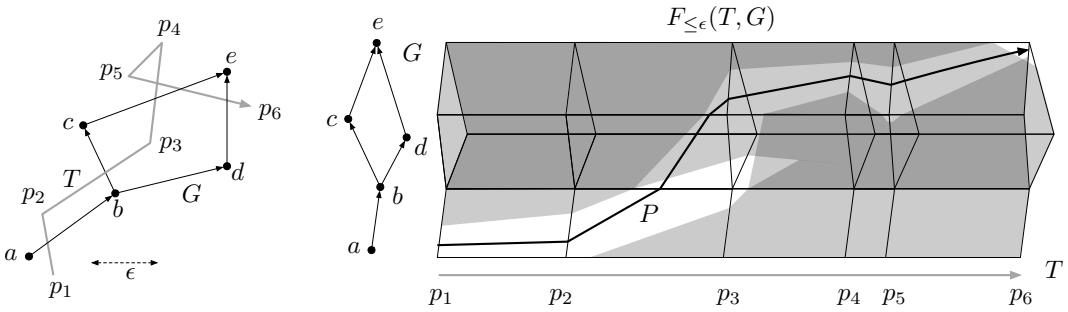

Figure 2 (left) A trace T in grey through the arcs of a graph G. (right) The free-space diagram with T stretched out horizontally and G in the vertical direction shows unreachable areas in dark color. Any vertex-monotone path through the reachable free-space, like the path P in black, we consider a valid map matching with distance at most ϵ. Note how the example is not monotone between p_4 and p_5 along arc (c, e). Also note how for this distance ϵ, there is a path through the reachable free-space that maps T to the path $a \to b \to c \to e$, but there is none along $a \to b \to d \to e$ as vertex d is too far away from the trace.

3 General Framework for Map Matching Algorithms

To motivate our algorithm, we introduce a general framework for map matching that includes more than just Fréchet distance map matching.

For a discrete GPS trace $T : [\![1, n]\!] \mapsto \mathbb{R}^2$, a map matching can be formalized as a function $M : [\![1, n]\!] \mapsto G$, where G is an embedded graph representing the road network. We observe that if we take the Cartesian product of any continuous relaxation $\tau : [1, n] \mapsto \mathbb{R}^2$ and $\mu : [1, n] \mapsto G$ of T and M, we get $\pi = \tau \times \mu : [1, n] \mapsto \mathbb{R}^2 \times G$. Moreover, any such $\rho = \tau \times \mu$ where $\tau(i) = T(i)$ where $i \in \mathbb{Z}$ yields a map matching $M : [\![1, n]\!] \mapsto G$ by restricting the domain of μ to integers. Therefore, it is natural to consider a class of map matching algorithms that optimize a cost function $\gamma(\rho)$ on functions $\rho = \tau \times \mu : [1, n] \mapsto \mathbb{R}^2 \times G$ where $\tau(i) = T(i)$. We claim that both classical Hidden Markov Model (HMM for short) and Fréchet map matching approaches fall into this class of algorithms.

3.1 Hidden Markov Model

The HMM algorithm by Newson and Krumm [32] is one of the most popular map matching algorithms, and is the basis of map matching implementations in libraries such as GraphHopper [24] and MapBox [34]. The general approach of this algorithm is to find matched roads $\{r_i\}_{i=1}^n$ that maximize

$$\Pr(T(n)|r_n) \prod_{i=1}^{n-1} \Pr(T(i)|r_i) \Pr(d_i). \tag{6}$$

In the original paper, these probabilities are defined with model parameters σ_z, β as

$$\Pr(T(i)|r_i) = \frac{1}{\sqrt{2\pi}\sigma_z} e^{-0.5(\|T(i)-x_i\|_2/\sigma_z)^2} \tag{7}$$

where x_i is the point on r_i closest to $T(i)$ and

$$\Pr(d_i) = \frac{1}{\beta} e^{-d_i/\beta} \tag{8}$$

where

$$d_i = \left| \|T(i) - T(i+1)\|_2 - \|x_i - x_{i+1}\|_2 \right|. \tag{9}$$

Note that maximizing (6) is equivalent to minimizing the following cost function:

$$\gamma(\rho) = -\sum_{i=1}^{n} \log \Pr(T(i)|r_i) - \sum_{i=1}^{n-1} \log \Pr(d_i). \tag{10}$$

This cost function can be computed from $\rho = \tau \times \mu$, as r_i is simply the road that $\mu(i)$ lies on, and

$$d_i = \left| \|\tau(i) - \tau(i+1)\|_2 - \int_i^{i+1} \nu_i(t)dt \right| \tag{11}$$

where $\nu_i(t) = \mu'(t)$ if μ traverses the shortest path on G from $\mu(i)$ to $\mu(i+1)$, and $\nu_i(t) = \infty$ otherwise.

Now, the problem can be viewed through the lens of finding the least costly path with respect to γ through $\mathbb{R}^2 \times G$. To make this efficient to optimize, the HMM algorithm restricts $\mu(i)$ to projected points of road within a small neighborhood of each $T(i)$. This discretization of the problem naturally suggests defining vertices in $(T(i), x_i) \in \mathbb{R} \times G$ with costs $-\log \Pr(T(i)|r_i)$ and shortcut edges of cost $-\log \Pr(d_i)$ between them and then using Dijkstra's algorithm for shortest paths to find the least costly path with respect to γ. The algorithm of Tang et al. [45], for example, behaves in a similar fashion, although it does not explicitly model the problem as a HMM.

3.2 Fréchet distance

Map matching by the Fréchet distance [4] also naturally falls in this framework. To see this, we simply plug in the definition of the Fréchet distance and note that minimizing the Fréchet distance is the same as minimizing the cost function

$$\gamma(\rho) = \max_{i \in [1,n]} \|\tau(i) - \mu(i)\|_2 \tag{12}$$

if τ is a monotonic parameterization of the input trace, and μ monotone on its projected path on the graph and $\gamma(\rho) = \infty$ otherwise. We note that it is this monotonicity condition that precludes a simple implementation of this path optimization problem using Dijkstra's algorithm [16] in map matching. Instead, classical algorithms for Fréchet map matching find optimal values for γ through binary or parametric search.

3.3 Motivation for our algorithm

With this framework in mind, our goal becomes finding a function γ that captures how "good" a map matching is, and that, at the same time, is efficient to optimize. We observe that if we use the vertex-monotone Fréchet distance instead of the regular Fréchet distance, we are optimizing (12) without global monotonicity constraints, and such an optimization can be implemented with a single Dijkstra's search. One issue with raw Fréchet map matching is that there may be many ties with the same γ. Wei et al. [48, 49] found that using a HMM-like objective function on matchings with the same Fréchet distance gave good map matching results, which inspired us to use a similar secondary optimization in our map matching algorithm. This secondary optimization, like the HMM, can be implemented using Dijkstra's algorithm, notably on an even smaller portion of the parameter space. Details for our algorithm are described in Section 4.

4 Algorithm

On a high level, our algorithm for vertex-monotone Fréchet distance map matching combines three existing ideas in a new way:

- We follow the general two-step approach of Wei et al. [48, 49] to first compute the necessary Fréchet distance for a match to exist and then to optimize the path within that distance threshold using a secondary objective function.
- We build a representation of the reachable free-space on the fly. This was not done by Wei et al. but by Wenk et al. [50] in the context of using the weak Fréchet distance.
- Finally, we introduce the vertex-monotone Fréchet distance metric to map matching to overcome the shortcomings of the weak Fréchet distance described in Section 2.1 and to significantly improve the running time compared to the standard Fréchet distance.

4.1 Overview

Our algorithm works in two steps: First, we determine the vertex-monotone Fréchet distance between T and G by running a *minimum bottleneck path* search through the free-space diagram of T and G. The only parameter of this search is a maximum distance D, which we use to decide whether there is a match for this trace or not.

Second, we search for the best path among all the ones with minimum distance. For this tie-breaking step, we use a global weight-function to trade off minimizing the length of the matching path P through the graph with minimizing the distances between each point $p_i = \tau(t)$ of T and its corresponding match $x_i = \mu(t)$ on G.

As pointed out by Wei et al. [48, 49], such a second step is necessary to differentiate among the often many paths P that have the same vertex-monotone Fréchet distance. Note that a single point with high GPS error and possibly large distance from a nearest road segment may significantly increase the number of possible segments for many other points. For instance, such a weight function allows us to prefer the straight main street over a nearby parallel parking road as it avoids a detour, even if the GPS signal of a car driving on the main street is off and some individual crumbs appear to be closer to the parking road than the main street (see Figure 3 for an example).

4.2 Distance computation

The first phase of our algorithm has two goals: Determining the vertex-monotone Fréchet distance $d^*_{\text{VMFD}} = \min_{P \in G} d_{\text{VMFD}}(P, G)$ as well as, if $d^*_{\text{VMFD}} \leq D$, exploring the entire reachable free-space of $F_{\leq d^*_{\text{VMFD}}}$.

4.2.1 Auxiliary interval graph

To this end, we traverse the free-space surface in increasing order of the vertex-monotone Fréchet distance. We look at this as a graph traversal problem, where the vertices are the border intervals of the cells of the free-space diagram. In particular, we look at *vertical intervals* $(p_i, (v_j, v_k))$, determined by a crumb $p_i \in T$ and an arc $(v_j, v_k) \in A$, and at *horizontal intervals* $((p_i, p_{i+1}), v_j)$, determined by two consecutive crumbs $p_i, p_{i+1} \in T$ and a vertex $v_j \in V$. Each interval naturally defines a distance, namely the distance between the point and the arc for vertical intervals and between the crumb line and the vertex for horizontal ones, that we use as a vertex weight in the interval graph.

To define the connectivity of this auxiliary interval graph, we consider all the options of a vertex-monotone parametrization. Vertical intervals $(p_i, (v_j, v_k))$ only have two options to continue: We can move from p_i to p_{i+1} to the next vertical interval $(p_{i+1}, (v_j, v_k))$,

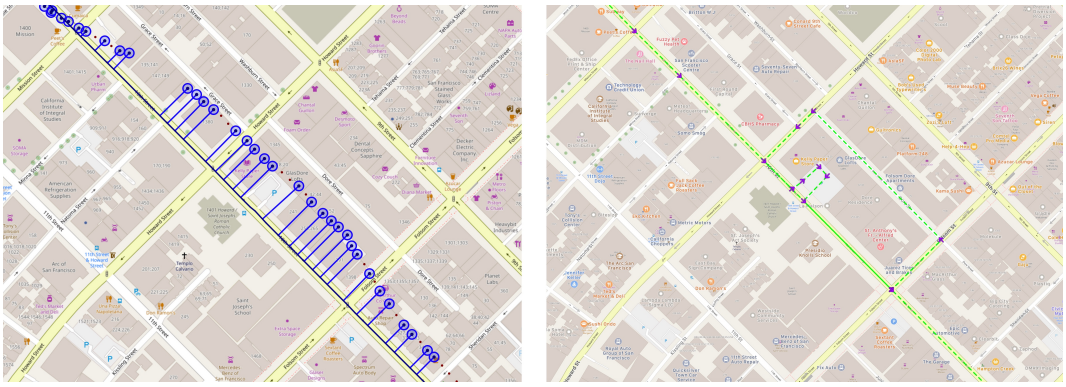

Figure 3 (left) A trace proceeding along 10th Street in San Francisco from northwest to southeast with its correctly map matched path computed with the vertex-monotone Fréchet distance map matching algorithm (with simplificiation enabled). Note how the GPS points are consistently shifted by about 45 meters to the northeast for the central part of the depicted trace. (right) The path computed by the GraphHopper HMM implementation (without simplification). Observe how the shifted points unsettle the algorithm and entice the path to enter into the parking lot and then to even go once around the entire block to travel through Dore Street. Dore Street is significantly closer to the observed GPS points and thus corresponds to a higher likelihood in the HMM. However, the much longer matching path and the fact that one has to use it in the opposing direction (as Howard Street and Folsom Street are one-way streets) make it clear that this is not the correct match. This example illustrates why we believe that the independence assumption for GPS errors often does not hold in practice and that, due to its geometric nature, our proposed algorithm is more robust to such consistent offsets.

which corresponds to matching the crumb line (p_i, p_{i+1}) to within the arc (v_j, v_k), or we can proceed to the end of the arc, so to the horizontal interval $((p_i, p_{i+1}), v_k)$, which corresponds to matching the end of (v_j, v_k) to within the interval (p_i, p_{i+1}). Horizontal intervals $((p_i, p_{i+1}), v_j)$ have two continuations for each outgoing arc of v_j: For an arc (v_j, v_k), we can go to another horizontal interval $((p_i, p_{i+1}), v_k)$, saying that we match all of (v_j, v_k) to within (p_i, p_{i+1}), or we can go to the vertical interval at the end of the crumb line, so to $(p_{i+1}, (v_j, v_k))$, saying that we match the end of (p_i, p_{i+1}) to within the arc (v_j, v_k).

Note how we do not add interval arcs for going to the previous crumb or to the beginning of an arc. Those would be the arcs needed to allow for searching the weak Fréchet distance. We refer to Figure 4 for an illustration of this interval graph on the free-space diagram and the possible out-arcs of each interval vertex.

4.2.2 Correctness

We now argue that computing the vertex-monotone Fréchet distance corresponds to finding a path with the smallest possible maximum vertex weight, the so-called *minimum bottleneck path*, through this interval graph starting at any vertical interval involving p_1 and ending at any vertical interval involving p_n. To prove this, we observe that for any two line segments in $\mathbb{R}^2$ the maximum distance between the two always involves at least one of the four endpoints. This means that for any cell of the free-space surface and any two points on neighboring or opposite border intervals of the cell (which thus define two line segments), there is always a monotone (but not necessarily increasing) parametrization within the cell, so that the maximum distance is at one of the two points. Hence we only need to worry about the

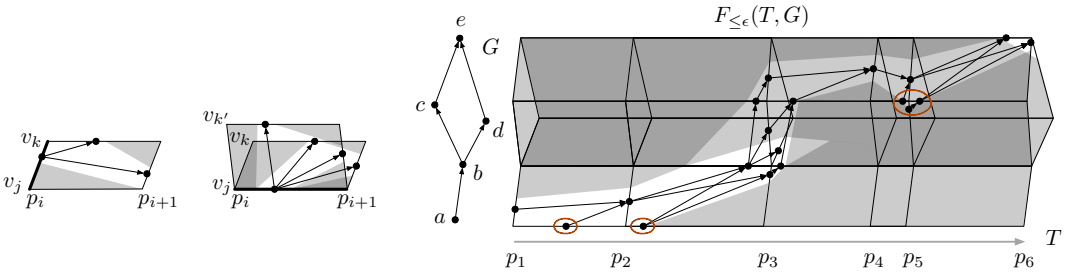

Figure 4 (left) A vertical and a horizontal interval of a free-space graph (in bold) with their vertex and their successors. We draw the vertex at the position within the interval that represents the point on the arc closest to the crumb, or the point on the crumb line closest to the vertex, respectively. Note how two of the arcs go backwards with respect to one of the parameters, so they correspond to non-monotone parametrizations. (right) The partial interval graph of the free-space diagram for a distance threshold ϵ. Note that there are multiple paths from $(p_1, (a, b))$ to $(p_5, (c, e))$ (differing around p_3) that each correspond to a vertex-monotone Fréchet distance map matching. Also note the five vertices circled in red that are in the free-space but are not reachable from the only starting vertex $(p_1, (a, b))$.

intersections of the parametrization with the grid lines of the free-space diagram. As vertex-monotonicity allows us to pick any parametrization point on an interval irrespective of the parametrization point on previous intervals, we can always pick the point that corresponds to the projection of the crumb onto the arc (for vertical intervals) or of the vertex onto the crumb line (for horizontal intervals), which is the interval weight we defined above.

4.2.3 Implementation

Let us now dive into the implementation of this bottleneck path search. We start by searching for all roads that are within distance D of the starting crumb p_1. Note that this is the only spatial query in our algorithm (unlike HMM, which searches for nearby roads for every crumb). For all these candidate roads, we build their vertical intervals and insert them into a priority queue keyed by the interval weight. We now run a modified version of Dijkstra's single-source shortest path algorithm [16] to find the minimum bottleneck path to any vertical interval containing p_n. Once we find such a path, we keep exploring the graph for as long as the bottleneck does not increase. If at any point the bottleneck reaches D, we abort the search.

It is important to note that we build the interval graph on the fly, i.e., we do not enumerate all interval vertices and arcs of the free-space diagram, but only traverse those reachable within distance d^*_{VMFD} from the start. We store all index triples of the reachable intervals in a hash set to be able to quickly constrain our later path optimization to this subgraph.

4.2.4 Running time

The running time of this distance computation depends on two things: the initial spatial index lookup and the bottleneck path search. We use a Geohash-based [33] hash-table lookup as the spatial index, which for a constant radius D looks up a constant number of hash-table entries and thus runs in time linear to the number of roads returned. In particular, the running time is independent of $|A|$, the size of the graph. The bottleneck path search runs in time $\mathcal{O}(n^* \log n^* + m^*)$, where n^* and m^* are the number of vertices and arcs of the free-space diagram $F_{\leq d^*_{\text{VMFD}}}$ being explored.

Note that while there are faster algorithms in the worst case, i.e., ways of shaving the log, for the minimum bottleneck path problem on undirected graphs (using linear-time median pivoting and shrinking of connected components, see [37]) and on directed acyclic graphs (processing the vertices in topological order voids the need for a priority queue), Djikstra's allows us easily to limit our search space on the graph, and give up if the bottleneck distance becomes too large.

It is also worth comparing our running time with that of the regular Fréchet distance. To observe the monotonicity constraint of the Fréchet distance, we can not always pick the point with the smallest distance along the interval of the free-space. In fact, the range of parametrization points that is feasible on an interval depends on the choice on the previous interval as well as on the final Fréchet distance that we target. Hence, when computing the Fréchet distance, one usually solves the decision problem, i.e., computes the earliest reachable point on each interval for a given ϵ, and then performs a binary or parametric search to find d^*_{FD}. For that decision problem no priority queue is needed, so the running time can be bounded by $\mathcal{O}(\log(D) \cdot (n' + m'))$ where n' and m' are the number of vertices and arcs of the free-space diagram of $F_{\leq D}$. To stress why computing the vertex-monotone Fréchet distance is much faster than the Fréchet distance, it is important to note that the running times for the two distance measures do not just replace the log of the binary search with the log of the priority queue, but that computing d^*_{VMFD} only involves looking at $F_{\leq d^*_{\mathrm{VMFD}}}$ while computing d^*_{FD} involves looking at $F_{\leq 2d^*_{\mathrm{FD}}}$ or even $F_{\leq D}$ depending on the implementation of the parameter search. As the complexity of the reachable free-space grows roughly quadratic with the search radius, overshooting the search radius when solving the decision problem can heavily influence the running time.

4.3 Path optimization

After determining d^*_{VMFD}, the second step of our algorithm is to select the best path P^* among all those with $d_{\mathrm{VMFD}}(T, P) = d^*_{\mathrm{VMFD}}$.

For this tie-breaking step, we minimize a global weight function inspired by Wei et al. [48, 49], namely

$$w(P) = \sum_{i=1}^{n} (\Delta_i + \Delta_{i+1}) \left\| x_i - p_i \right\|_2 + \alpha \sum_{i=1}^{n-1} l_i, \tag{13}$$

where α is a constant, $\Delta_i = \left\| p_{i-1} - p_i \right\|_2$, x_i is the point in the graph where p_i gets matched to and l_i is the length of the matched path between x_i and x_{i+1}. For the sentinel cases, we use $\Delta_1 = \Delta_{n+1} = \beta$, for some constant β.

The first sum in $w(P)$ measures the closeness of the matched path to the trace, trying to encourage P to follow T not just at the extreme point (where d^*_{VMFD} is determined) but also everywhere else. By weighing each crumb matching distance with the distance to the previous and next crumb, we weigh those crumbs higher where the user was moving faster or the sampling frequency is lower, which makes this formula independently of the sampling rate (similar to the t_i term by Wei et al.).

The second sum of $w(P)$ measures the length of the matching path P, discouraging P from taking local detours in areas where the free-space allows multiple options. We found that setting $\alpha = D$ and $\beta = 8D$ works well, basically reducing the number of parameters of our whole algorithm to a single one D, which can easily be set based on the maximum GPS error one expects to see in T.

One intuition for why we like this weight formula is that in the simple case where T and P are both straight horizontal lines with a constant vertical offset of a "typical" GPS error of $\frac{D}{2}$ between them, both summands correspond to (roughly) twice the size of the area of

the rectangle spanned by T and P independent of the sampling rate or driving speed. While the qualities of such a weight function are undeniably subjective, we argue that ours is a slightly more natural way to trade off trace proximity and path length than the one by Wei et al., especially since it involves the same units (meters squared) on both sides of the sum.

4.3.1 Shortest path interpretation

This weight $w(P)$ can be optimized using a shortest-path computation on the reachable free-space of $F_{\leq d^*_{\mathrm{VMFD}}}$. The summands $(\|p_{i-1} - p_i\|_2 + \|p_{i+1} - p_i\|_2)\,\|x_i - p_i\|_2$ correspond one-to-one to vertex weights for all the n vertical intervals on every P. The summands for $\alpha \cdot l_i$ can be split up to arc weights on the interval graph, where each interval arc measures the progress (forward or backward) along the graph arc involved. As an example, an interval arc from one horizontal interval to another has to account for the entire length of the arc in the graph. Using the hash set of reachable intervals from the first step, we can again use Dijkstra's single-source shortest path algorithm to find P^* as well as all the x_i in time $\mathcal{O}(n^* \log n^* + m^*)$. Note that in our implementation, we do not use Fibonacci heaps for the priority queue, so the asymptotic running time of our implementation is in fact $\mathcal{O}(n^* + m^* \log n^*)$.

4.4 Trace simplification

In this section, we describe an additional, optional speed-up technique in our implementation: if the temporal and spatial information in the trace T indicates that the user was travelling at a constant speed and direction for several crumbs, we can reduce the complexity of the free-space diagram by subsampling the trace before invoking the map-matching algorithm. If we then interpolate the resulting map matching, we can expect the upsampled x_i on P to closely match the positions we would have gotten when map matching the original trace.

There are many well-known curve simplification algorithms and the addition of a time component is straightforward for many of them. In particular, for a simplification threshold of 1 meter, we experimented with the algorithm by Ramer [40], Douglas and Peucker [17], the one by Reumann and Witkam [41] and a dynamic programming approach to optimally select the minimum number of points. In the end, we found the following approach to give a very good trade-off between the time spent simplifying the trace and the time saved in map matching (which closely correlates to the number of crumbs dropped). Our algorithm closely resembles the algorithm FrechetSimp by Agarwal et al. [1, Section 3.1], but instead of preserving the Fréchet distance, we also preserve the speed of the trace, which can be important in many applications. We found that simplification with a threshold of just one meter at any provided timestamp resulted in a significant reduction in runtime.

4.4.1 Doubling-search simplification

This algorithm scans through the trace from beginning to end and incrementally decides which crumbs to keep and which ones to skip. It does so with a slow-start binary search, i.e., it first tries skipping 1, then 2, then 4, then 8, ..., points until it fails (i.e., at least one of the skipped crumbs would be interpolated more than 1 meter away from its original position) and then uses regular binary search to find how many points can be skipped. Note that the predicate "can x points be skipped?" is not monotone in x, meaning that it might be possible to skip 6 points but not 5 points. In that sense, this algorithm is doing a best effort but is not guaranteed to greedily skip as many points as possible. What is guaranteed though is that if for some x we have that for all $y \leq x$, "y points can be skipped" holds, then the algorithm will skip at least x points (it might get lucky and skip even more points).

The running time of this algorithm is $\mathcal{O}(n \log n)$: The cost for checking whether x points can be skipped is $\mathcal{O}(x)$ and whenever we skip y points, the cost for doing so is bounded by the slow-start in $\mathcal{O}(1) + \mathcal{O}(2) + \mathcal{O}(4) + \cdots + \mathcal{O}(y) + \mathcal{O}(2y) = \mathcal{O}(y)$ and the binary search in $\mathcal{O}(y \log y)$. Note that this worst case only occurs if a long stretch of points can be skipped (in which case it is offset by time savings in the map matcher). However, the running time drops to linear if for example at least every tenth point has to be taken. We refer to Figure 5 for illustrations of this trace simplification.

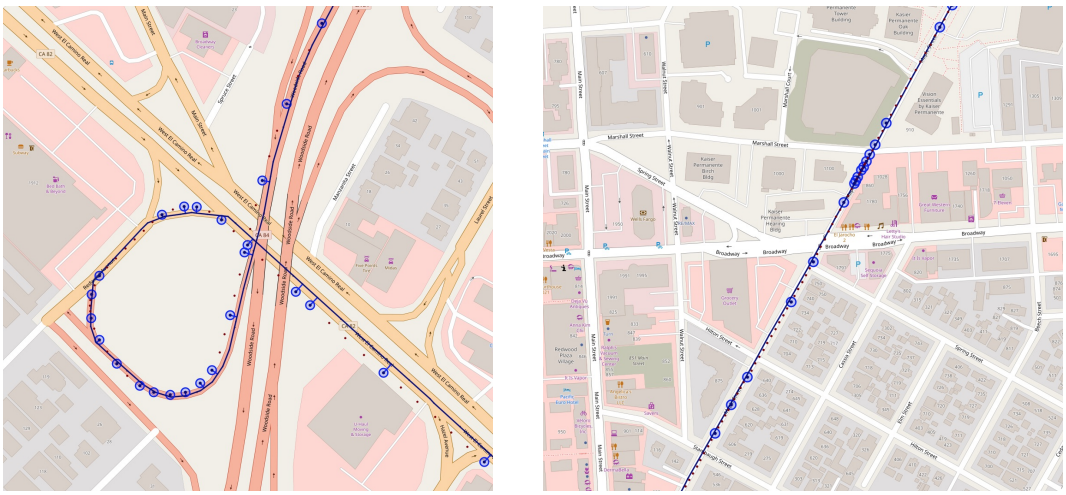

Figure 5 (left) This shows the spatial component of our trace simplification. On the straight sections many crumbs get skipped, while in the curved parts of the loop most crumbs are kept to ensure an accurate interpolation. (right) This shows the temporal component of our trace simplification. While the entire trace shown is a straight line, the driver had to stop at the intersection with Marshall Street. Therefore, many crumbs are retained there, whereas up to ten crumbs are skipped in the area where the car was driving at a uniform speed.

5 Experiments

Table 1 Experimental results: map matching times normalized to microseconds per crumb (shown as mean ± standard deviation). Lower is better. The algorithm leveraging vertex-monotone Fréchet distance (VMFD) is roughly 4 times faster than the regular Fréchet distance (FD) implementation, and one to two orders of magnitude faster than the GraphHopper HMM implementation. The addition of our geometric trace simplification method (columns marked with +S) results in a roughly two-fold speedup for the Fréchet distance-based methods, while only a small improvement can be observed for the GraphHopper HMM.

	Map matching time in μs per crumb					
	VMFD		FD		HMM	
Data set	+S		+S		+S	
SF Bay Area	1.9 ± 3.5	4.4 ± 7.1	6.8 ± 11	19.2 ± 31	519	662
Hong Kong	1.2 ± 2.2	2.3 ± 5.4	4.0 ± 10	8.4 ± 18	51	60

The main focus of our experimental evaluation is on performance. While processing times for a single trace matter, for most of our applications throughput matters most. In the following, we are reporting processing times normalized by crumb for two data sets, each with thousands of traces and millions of crumbs.

Table 2 Experimental results for the Bay Area data set for different maximum (vertex-monotone) Fréchet distances D: (left) Map matching times normalized to microseconds per crumb (shown as mean ± standard deviation). Lower is better. (right) Number of matching traces (out of 7736). The vertex-monotone Fréchet distance (VMFD) consistently outperforms the regular Fréchet distance (FD) by roughly a factor of 4, both with (+S) and without simplification.

	Map matching time in μs per crumb				Number of matching traces			
	VMFD		FD		VMFD		FD	
D	+S		+S		+S		+S	
4m	0.5 ± 0.5	0.4 ± 0.5	0.9 ± 1.3	1.0 ± 1.2	74	68	72	66
8m	0.6 ± 5.3	0.5 ± 0.6	1.3 ± 2.1	1.5 ± 1.9	407	408	403	402
16m	0.8 ± 5.8	0.9 ± 1.3	2.2 ± 4.0	3.3 ± 4.9	2297	2298	2287	2288
32m	1.2 ± 5.6	1.9 ± 1.3	4.0 ± 6.5	7.9 ± 11	4072	4067	4057	4051
64m	1.9 ± 3.5	4.4 ± 7.1	6.8 ± 11	19.2 ± 31	5467	5463	5456	5454
128m	3.6 ± 5.7	11.5 ± 24	13.1 ± 26	48.8 ± 102	6218	6216	6217	6215
256m	10.3 ± 23	38.3 ± 86	39.2 ± 104	166 ± 405	6875	6873	6875	6873

Table 3 Experimental results for the Hong Kong data set: (left) Map matching times normalized to microseconds per crumb (shown as mean ± standard deviation). Lower is better. (right) Number of matching traces (out of 3889). The vertex-monotone Fréchet distance (VMFD) consistently outperforms the regular Fréchet distance (FD) by roughly a factor of 4, both with (+S) and without simplification.

	Map matching time in μs per crumb				Number of matching traces			
	VMFD		FD		VMFD		FD	
D	+S		+S		+S		+S	
4m	0.4 ± 0.4	0.3 ± 0.4	0.7 ± 0.9	0.8 ± 0.9	10	8	10	8
8m	0.4 ± 0.4	0.3 ± 0.4	0.9 ± 1.3	1.2 ± 1.7	80	81	79	80
16m	0.5 ± 0.6	0.5 ± 0.7	1.3 ± 2.3	1.8 ± 3.0	377	376	377	376
32m	0.7 ± 1.0	1.0 ± 1.8	2.2 ± 4.8	3.6 ± 7.2	760	759	759	758
64m	1.2 ± 2.2	2.3 ± 5.4	4.0 ± 10	8.4 ± 18	1139	1137	1135	1133
128m	2.7 ± 5.6	6.2 ± 11	10.4 ± 29	27.3 ± 59	1611	1606	1610	1605
256m	7.5 ± 15	22.9 ± 49	30.0 ± 78	108 ± 256	2186	2178	2183	2175

5.1 Data sets

We used two sets of traces for our performance experiments. Both sets consist of OpenStreetMap traces that we downloaded using JOSM [29]. We preprocessed both trace sets by splitting traces at any gap of more than 15 seconds to ensure a high sampling rate and by only keeping traces with at least 60 crumbs so that they provide reasonable context for disambiguation. The first set consists of 7736 traces in the San Francisco Bay Area with a total of 3.62M crumbs. The second set consists of 3889 traces in Hong Kong, totaling at 1.63M crumbs. For the maps in our experiment, we extracted the street network out of OpenStreetMap osm.pbf files [22, 38] using RoutingKit [14, 15]. Note that this extraction step eliminates non-drivable segments such as hiking paths. The map of Northern California contains 11.7M vertices and 23.6M arcs, and the map of Hong Kong contains 280k vertices and 470k arcs.

5.2 Algorithms

We compare three algorithms, all implemented in Java:

1. vertex-monotone Fréchet distance map matching algorithm as described in Section 4 with $D = 64$ meters (and varying D later)
2. Fréchet distance map matching algorithm with the same maximum distance $D = 64$ meters
3. HMM algorithm [32] in the open-source library GraphHopper [24] (default settings)

We ran all experiments on a MacBook Pro from 2019 with a 2.3 GHz 8-core Intel Core i9 CPU and 32 GB RAM. Our implementation is single threaded, however, so only one core is being used. We ran all algorithms 3 times in a row and used the measurements of the last run.

5.3 Quality

We tested our algorithm on the ACM GIS Cup 2012 [3] data set, with corrections from Wei et al. [48, 49]. We disabled preprocessors and simplification, as the data set required a matched road for each crumb. Our algorithm resulted in a 97.75% match rate, which is in line with match rates reported in [3, 48, 49]. We visually inspected the differences from the provided ground truth, and they appeared to be ambiguous from a geometric standpoint. If certain modeling assumptions are desired, the optimization function can be modified to include them, and we leave that open for future work. The relaxation to vertex monotonicity can sometimes alter map matching results at the end points of segments, for example when the GPS signals fluctuate back and forth for a few meters for cars waiting at traffic lights, but the sequence of road segments matched does not change.

We also noticed that in some scenarios, such as shifted GPS locations in urban canyons, using a Fréchet distance type distance as our primary objective function has significant advantages over relying on distributional assumptions such as those in a HMM. Error distributions may not be independent, and modeling assumptions that rely on independence can result in unexpected map matching results, as for example in Figure 3.

5.4 Performance results

For an overview of experimental results, see Table 1.

Without trace simplification, we measured $4.4\mu s$ per crumb for our algorithm and $662\mu s$ per crumb for the GraphHopper HMM on the Bay Area traces. On the OSM Hong Kong traces, we measured $2.3\mu s$ per crumb for our algorithm and $60\mu s$ per crumb for the GraphHopper HMM. Therefore, our implementation is one to two orders of magnitude faster than the GraphHopper HMM.

For the regular Fréchet distance, we implemented a search procedure using a slow-start binary search up to at most 64 meters and down to a precision of 1 meter. We found the vertex-monotone Fréchet distance implementation to be roughly 4 times faster than the regular Fréchet distance implementation.

We also measured the impact of the trace simplification described in Section 4.4 and report a roughly two-fold speedup for an accuracy of 1 meter which results in about 2 out of 3 crumbs being dropped. Note that both vertex-monotone Fréchet distance and the regular Fréchet distance benefit from simplification. For completeness, we also ran the GraphHopper HMM on the simplified traces. As the simplification is a geometric simplification, both Fréchet distance map matching variants are fairly stable, whereas there is less reason to

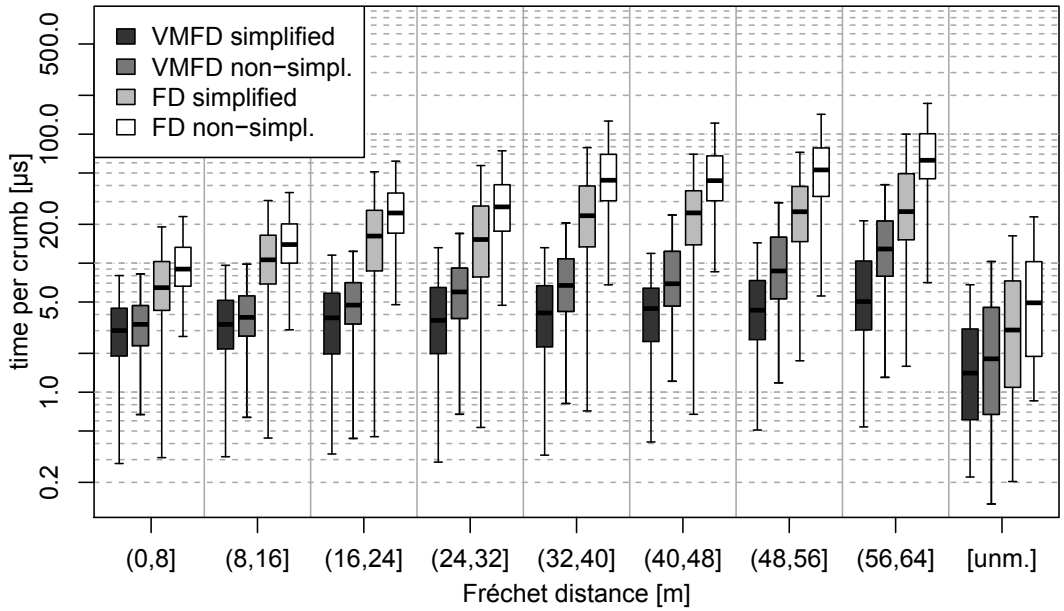

Figure 6 Box plots of the running times per crumb for the traces in the Bay Area dataset grouped by their (vertex-monotone) Fréchet distance. The last column [unm.] contains the traces that did not match under the maximum distance of $D = 64$ meters. We observe that the running time per crumb grows relatively slowly with increasing Fréchet distance. Furthermore, the traces that do not match are faster to process than the ones that do.

believe that the HMM model would be stable under this simplification. As can be seen in Table 1, it also turned out that the GraphHopper HMM does not benefit much from simplification, most likely because the overall lengths of paths being computed does not get reduced by much.

Finally, we also studied the effect of decreasing and increasing the maximum Fréchet distance parameter D from our default value of 64 meters. As Tables 2 and 3 show, the map matching time grows superlinear with D while the number of additional matched traces does not. Especially in the Bay Area data set, where most traces already match with 32 or 64 meters, increasing to larger D is not worth the extra computational effort in many applications. Manual inspection of the additionally matched traces with $D > 64$ meters showed that these are almost exclusively traces of hiking, biking, ferry, cablecar or train trips which therefore justifiably do not match against the street network at $D = 64$ meters. We also provide insight into the distribution of processing times by Fréchet distance in Figures 6 and 7.

5.5 Literature comparison

While run on different graphs, traces and computers, and possibly implemented with different programming languages, some previous papers report throughput numbers in their experimental sections, which we list here.

- For a sampling interval of 1 second, Wei et al. [48, 49] report a map matching time of around 400 000μs per crumb for their Fréchet distance implementation and of about 1000μs per crumb for the HMM by Newson and Krumm [32] (numbers from [48, page 7, logarithmic plot]).

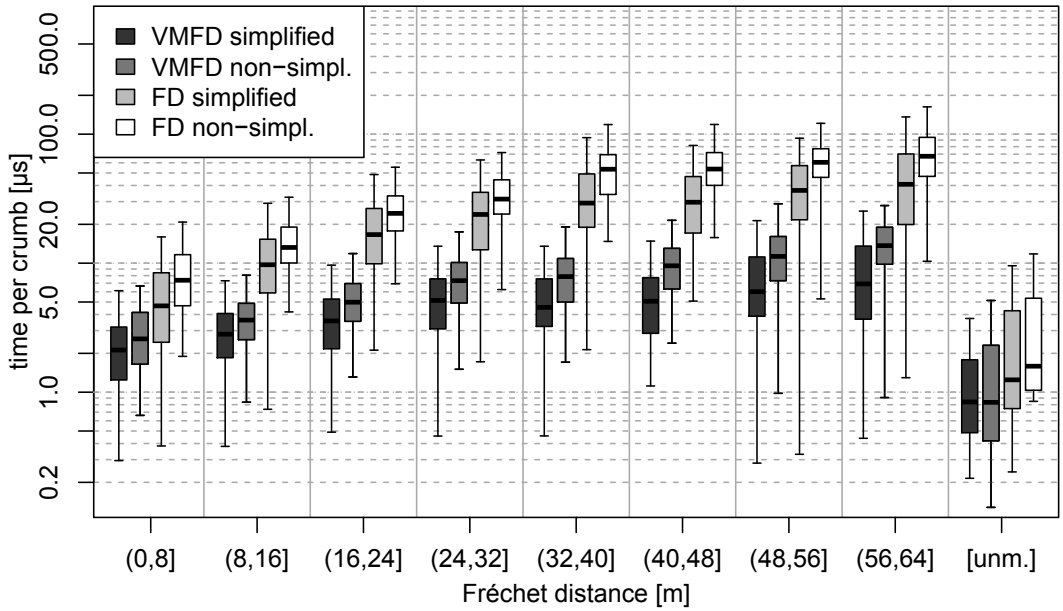

Figure 7 Box plots of the running times per crumb for the traces in the Hong Kong dataset grouped by their (vertex-monotone) Fréchet distance. The last column [unm.] contains the traces that did not match under the maximum distance of $D = 64$ meters. Same observation as for Figure 6.

- Tang et al. [45], the winners of the SIGSPATIAL cup 2012 on map matching, report a time of about $67\mu s$ per crumb [45, page 4, Figure 2].
- Finally, Yang and Gidofalvi [51] measured their optimized HMM at 22 to $40\mu s$ per crumb.

Many of these algorithms are implemented in C++, which allows for significantly more optimization when compared with our Java implementation. Even so, our algorithm exceeds previously reported throughputs by an order of magnitude and with its geometric guarantees is a strong alternative to HMM-based algorithms.

6 Conclusion

We propose the metric of vertex-monotone Fréchet distance as an effective alternative for efficient map matching. We show that Fréchet distance-based map matching can be fast even while combining the power of geometric optimization with global weight methods. Our algorithm requires very little parameter tuning and does not make strong assumptions on the distribution of GPS errors.

References

1 Pankaj K Agarwal, Sariel Har-Peled, Nabil H Mustafa, and Yusu Wang. Near-linear time approximation algorithms for curve simplification. *Algorithmica*, 42(3-4):203–219, 2005.

2 Mahmuda Ahmed, Sophia Karagiorgou, Dieter Pfoser, and Carola Wenk. A comparison and evaluation of map construction algorithms using vehicle tracking data. *GeoInformatica*, 19(3):601–632, 2015.

3 Mohamed Ali, John Krumm, Travis Rautman, and Ankur Teredesai. ACM SIGSPATIAL GIS cup 2012. In *20th International Conference on Advances in Geographic Information Systems, SIGSPATIAL'12*, pages 597–600, 2012.

4 Helmut Alt, Alon Efrat, Günter Rote, and Carola Wenk. Matching planar maps. *J. Algorithms*, 49(2):262–283, 2003. Announced at SODA'03.

5 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5:75–91, 1995.

6 Javed A. Aslam, Sejoon Lim, Xinghao Pan, and Daniela Rus. City-scale traffic estimation from a roving sensor network. In *10th ACM Conference on Embedded Network Sensor Systems, SenSys'12*, pages 141–154. ACM, 2012.

7 Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk. On map-matching vehicle tracking data. In *31st International Conference on Very Large Data Bases, VLDB'05*, pages 853–864. ACM, 2005.

8 Lili Cao and John Krumm. From GPS traces to a routable road map. In *17th International Symposium on Advances in Geographic Information Systems, SIGSPATIAL'09*, pages 3–12. ACM, 2009.

9 Pablo Samuel Castro, Daqing Zhang, and Shijian Li. Urban traffic modelling and prediction using large scale taxi GPS traces. In *10th International Conference on Pervasive Computing, Pervasive'12*, volume 7319 of *Lecture Notes in Computer Science*, pages 57–72. Springer, 2012.

10 Erin W. Chambers, Brittany Terese Fasy, Yusu Wang, and Carola Wenk. Map-matching using shortest paths. *ACM Transactions on Spatial Algorithms and Systems*, 6(1):6:1–6:17, 2020.

11 Daniel Chen, Anne Driemel, Leonidas J. Guibas, Andy Nguyen, and Carola Wenk. Approximate map matching with respect to the Fréchet distance. In *14th Workshop on Algorithm Engineering and Experiments, ALENEX'11*, pages 75–83, 2011.

12 Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable route planning in road networks. *Transportation Science*, 51(2):566–591, 2017.

13 Daniel Delling, Dennis Schieferdecker, and Christian Sommer. Traffic-aware routing in road networks. In *34th IEEE International Conference on Data Engineering, ICDE'18*, pages 1543–1548, 2018.

14 Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable contraction hierarchies. *ACM J. Exp. Algorithmics*, 21(1):1.5:1–1.5:49, 2016.

15 Julian Dibbelt, Ben Strasser, and Dorothea Wagner. RoutingKit. `https://github.com/RoutingKit/RoutingKit`, 2020.

16 Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

17 David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the international journal for geographic information and geovisualization*, 10(2):112–122, 1973.

18 Gregory D. Erhardt, Sneha Roy, Drew Cooper, Bhargava Sana, Mei Chen, and Joe Castiglione. Do transportation network companies decrease or increase congestion? *Science Advances*, 5(5), 2019.

19 Nivan Ferreira, Jorge Poco, Huy T. Vo, Juliana Freire, and Cláudio T. Silva. Visual exploration of big spatio-temporal urban data: A study of new york city taxi trips. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2149–2158, 2013.

20 M Maurice Fréchet. Sur quelques points du calcul fonctionnel. *Rendiconti del Circolo Matematico di Palermo (1884-1940)*, 22(1):1–72, 1906.

21 Stefan Funke, Tobias Rupp, André Nusser, and Sabine Storandt. PATHFINDER: storage and indexing of massive trajectory sets. In *16th International Symposium on Spatial and Temporal Databases, SSTD'19*, pages 90–99. ACM, 2019.

22 Geofabrik GmbH. Geofabrik OSM NorCal map. `https://download.geofabrik.de/north-america/us/california/norcal.html`, 2020.

23 Chong Yang Goh, Justin Dauwels, Nikola Mitrovic, Muhammad Tayyab Asif, Ali Oran, and Patrick Jaillet. Online map-matching based on hidden Markov model for real-time traffic sensing applications. In *15th International IEEE Conference on Intelligent Transportation Systems, ITSC'12*, pages 776–781, 2012.

24 GraphHopper. Map matching based on graphhopper. `https://github.com/graphhopper/map-matching`, 2020.

25 Songtao He, Favyen Bastani, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, and Sam Madden. Roadrunner: improving the precision of road network inference from GPS trajectories. In *26th International Conference on Advances in Geographic Information Systems, SIGSPATIAL'18*, pages 3–12. ACM, 2018.

26 Abdeltawab M. Hendawi, Sree Sindhu Sabbineni, Jianwei Shen, Yaxiao Song, Peiwei Cao, Zhihong Zhang, John Krumm, and Mohamed H. Ali. Which one is correct, the map or the GPS trace. In *27th International Conference on Advances in Geographic Information Systems, SIGSPATIAL'19*, pages 472–475. ACM, 2019.

27 Roel Jacobs. Constructing maps by clustering trajectories. Master's thesis, TU Eindhoven, 2016.

28 Erik Jenelius and Haris N. Koutsopoulos. Travel time estimation for urban road networks using low frequency probe vehicle data. *Transportation Research Part B: Methodological*, 53:64–81, 2013.

29 JOSM. An extensible editor for openstreetmap. `https://josm.openstreetmap.de`, 2020.

30 Benjamin B. Krogh, Christian S. Jensen, and Kristian Torp. Efficient in-memory indexing of network-constrained trajectories. In *24th ACM International Conference on Advances in Geographic Information Systems SIGSPATIAL'16*, pages 17:1–17:10. ACM, 2016.

31 Yang Li, Qixing Huang, Michael Kerber, Lin Zhang, and Leonidas J. Guibas. Large-scale joint map matching of GPS traces. In *21st International Conference on Advances in Geographic Information Systems, SIGSPATIAL'13*, pages 214–223. ACM, 2013.

32 Paul Newson and John Krumm. Hidden Markov map matching through noise and sparseness. In *17th ACM International Symposium on Advances in Geographic Information Systems, SIGSPATIAL'09*, pages 336–343. ACM, 2009.

33 Gustavo Niemeyer. geohash.org is public! `https://web.archive.org/web/20080305223755/http://blog.labix.org/#post-85`, 2008.

34 Patrick Niklaus. Matching GPS traces to a map. `https://blog.mapbox.com/matching-gps-traces-to-a-map-73730197d0e2`, 2015.

35 Bei Pan, Yu Zheng, David Wilkie, and Cyrus Shahabi. Crowd sensing of traffic anomalies based on human mobility and social media. In *21st International Conference on Advances in Geographic Information Systems, SIGSPATIAL'13*, pages 334–343. ACM, 2013.

36 Gang Pan, Guande Qi, Wangsheng Zhang, Shijian Li, Zhaohui Wu, and Laurence Tianruo Yang. Trace analysis and mining for smart cities: issues, methods, and applications. *IEEE Communications Magazine*, 51(6), 2013.

37 Matthias Peinhardt and Volker Kaibel. On the bottleneck shortest path problem. Technical report, Technical Report ZIB-Report 06-22, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 2006.

38 OpenStreetMap project. OSM extract Hong Kong. `http://download.openstreetmap.fr/extracts/asia/china/`, 2020.

39 Mohammed A. Quddus, Washington Y. Ochieng, and Robert B. Noland. Current map-matching algorithms for transport applications: State-of-the art and future research directions. *Transportation Research Part C: Emerging Technologies*, 15(5):312–328, 2007.

40 Urs Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer graphics and image processing*, 1(3):244–256, 1972.

41 K Reumann and APM Witkam. Optimizing curve segmantation in computer graphics. In *International Computing Symposium 1973*, pages 467–472, 1974.

42 Stefan Schrödl, Kiri Wagstaff, Seth Rogers, Pat Langley, and Christopher Wilson. Mining GPS traces for map refinement. *Data Mining and Knowledge Discovery*, 9(1):59–87, 2004.

43 Dominik Schultes and Peter Sanders. Dynamic highway-node routing. In *6th International Workshop on Experimental Algorithms, WEA'07*, volume 4525 of *Lecture Notes in Computer Science*, pages 66–79. Springer, 2007.

44 Junichi Shigezumi, Tatsuya Asai, Hiroaki Morikawa, and Hiroya Inakoshi. A fast algorithm for matching planar maps with minimum Fréchet distances. In *4th International ACM Workshop on Analytics for Big Geospatial Data, BigSpatial@SIGSPATIAL'15*, pages 25–34, 2015.

45 Youze Tang, Andy Diwen Zhu, and Xiaokui Xiao. An efficient algorithm for mapping vehicle trajectories onto road networks. In *International Conference on Advances in Geographic Information Systems, SIGSPATIAL'12*, pages 601–604. ACM, 2012.

46 Rolf van Leusden. A novel algorithm for computing the Fréchet distance. Master's thesis, TU Eindhoven, 2013.

47 Yilun Wang, Yu Zheng, and Yexiang Xue. Travel time estimation of a path using sparse trajectories. In *20th ACM International Conference on Knowledge Discovery and Data Mining, KDD'14*, pages 25–34. ACM, 2014.

48 Hong Wei, Yin Wang, George Forman, and Yanmin Zhu. Map matching by Fréchet distance and global weight optimization. *Technical Paper*, page 19, 2013.

49 Hong Wei, Yin Wang, George Forman, and Yanmin Zhu. Map matching: comparison of approaches using sparse and noisy data. In *21st International Conference on Advances in Geographic Information Systems, SIGSPATIAL'13*, pages 434–437. ACM, 2013.

50 Carola Wenk, Randall Salas, and Dieter Pfoser. Addressing the need for map-matching speed: Localizing global curve-matching algorithms. In *18th International Conference on Scientific and Statistical Database Management, SSDBM'06*, pages 379–388, 2006.

51 Can Yang and Gyozo Gidofalvi. Fast map matching, an algorithm integrating hidden Markov model with precomputation. *International Journal of Geographical Information Science*, 32(3):547–570, 2018.

52 Yu Zheng. Trajectory data mining: An overview. *ACM Transactions on Intelligent Systems and Technology*, 6(3):29:1–29:41, 2015.

53 Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. Mining interesting locations and travel sequences from GPS trajectories. In *18th International Conference on World Wide Web, WWW'09*, pages 791–800. ACM, 2009.