







Map-Matching Queries Under Fréchet Distance on Low-Density Spanners

Kevin Buchin   

Department of Computer Science, TU Dortmund, Germany

Maike Buchin   

Faculty of Computer Science, Ruhr-Universität Bochum, Germany

Joachim Gudmundsson   

School of Computer Science, University of Sydney, Australia

Aleksandr Popov   

Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Sampson Wong   

Department of Computer Science, University of Copenhagen, Denmark

Abstract

Map matching is a common task when analysing GPS tracks, such as vehicle trajectories. The goal is to match a recorded noisy polygonal curve to a path on the map, usually represented as a geometric graph. The Fréchet distance is a commonly used metric for curves, making it a natural fit. The map-matching problem is well-studied, yet until recently no-one tackled the data structure question: preprocess a given graph so that one can query the minimum Fréchet distance between all graph paths and a polygonal curve. Recently, Gudmundsson, Seybold, and Wong [13] studied this problem for arbitrary query polygonal curves and c -packed graphs. In this paper, we instead require the graphs to be λ -low-density t -spanners, which is significantly more representative of real-world networks. We also show how to report a path that minimises the distance efficiently rather than only returning the minimal distance, which was stated as an open problem in their paper.

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry

Keywords and phrases Map Matching, Fréchet Distance, Data Structures

Digital Object Identifier 10.4230/LIPIcs.SoCG.2024.27

Funding *Aleksandr Popov*: Supported by the Dutch Research Council (NWO) under the project number 612.001.801.

Sampson Wong: Supported in part by Starting Grant 1054-00032B from the Independent Research Fund Denmark under the Sapere Aude research career programme.

1 Introduction

Location data is ubiquitous, and analysis of that data is a common task. GPS trajectories of vehicles or people often suffer from being noisy or skipping large portions of the movement. To analyse them more precisely, one may use *map matching*. The idea is that the vehicles move on a road network, so one could snap their trajectories to a road network in a way that most closely resembles the original. We assume that the trajectory is a polygonal curve and the map is a geometric graph in the plane.

The map-matching problem has received considerable attention, with multiple surveys comparing the various approaches on different types of data [1, 7, 15, 16, 24, 25]. The *Fréchet distance* is a natural measure of similarity for polygonal curves [3, 11], taking into account the ordering of the points of the polygonal curves and capturing the maximal distance along them. There is a host of work considering map matching specifically under the Fréchet



© Kevin Buchin, Maike Buchin, Joachim Gudmundsson, Aleksandr Popov, and Sampson Wong;

licensed under Creative Commons License CC-BY 4.0

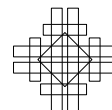
40th International Symposium on Computational Geometry (SoCG 2024).

Editors: Wolfgang Mulzer and Jeff M. Phillips; Article No. 27; pp. 27:1–27:15



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



distance [2, 5, 6, 8, 9, 14, 22], including a seminal paper by Alt, Efrat, Rote, and Wenk [2]. Their algorithm requires $\mathcal{O}(mn \log mn \log n)$ time and $\mathcal{O}(mn)$ space to match a polygonal curve of length m to a planar graph $G = (V, E)$ with complexity $|V| + |E| = n$. As shown via a conditional lower bound by Gudmundsson, Seybold, and Wong [13], this query time is close to optimal for planar graphs: there is no algorithm that runs in $\mathcal{O}((mn)^{1-\delta})$ time for any $\delta > 0$ that solves this problem after polynomial-time preprocessing of the graph. However, real-world road networks are rarely planar due to the presence of bridges and tunnels, so we would like to find other assumptions on the graph that also allow for faster query times.

Chen, Driemel, Guibas, Nguyen, and Wenk [8] study the map-matching problem under *realistic input* assumptions, which aim to exclude particular types of degenerate instances to provide stronger results. In their work in particular, the graph has low density and the trajectory is c -packed. A polygonal curve is called c -packed if in any ball of radius r , the total length of the curve inside the ball is at most cr . We can use a similar definition for geometric graphs by measuring the total length of the edges inside the ball. Unfortunately, c -packedness is a strong assumption that is difficult to satisfy. We instead define low-density graphs, which are more representative of real-world networks.

► **Definition 1.** *A geometric graph $P = (V, E)$ is λ -low density [21, 23] if for every disk of radius $r > 0$ in the plane, there are at most λ edges of length at least $2r$ intersecting the disk.*

Our work has no assumptions on the trajectories at the expense of stricter assumptions on the maps. Most often one will have a large number of trajectories being mapped to a relatively complex network, so to avoid the steep dependency on network complexity when matching every trajectory, we consider the query version of the problem. We preprocess the map so that we can quickly answer many map-matching queries, where each query is a trajectory. To our knowledge, this problem has only been studied on c -packed graphs [13, 14]. Gudmundsson and Smid [14] show an approach for c -packed trees with long edges and query trajectories with long edges. Gudmundsson et al. [13] assume that the graph is c -packed, but do not impose any restrictions on the query trajectories. However, c -packedness is not a realistic assumption for graphs representing road networks. Consider the example map of Figure 1: it is not c -packed for any constant c , as that would require the total length of the roads be at most cr in all disks of radius r , and it is instead often much closer to cr^2 . On some scale, this problem arises with many road networks, including city streets or motorways. Therefore, we would like to devise an approach with more realistic assumptions on the graph.

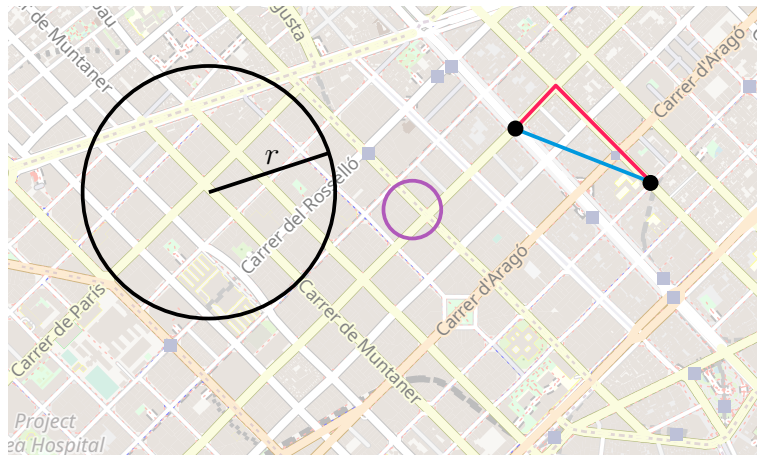
We instead assume that our graph has low density, defined above. As observed by Chen et al. [8], the value of density does not grow with the considered area on the map, and road networks typically have low density; it is also a strictly weaker assumption than packedness. We further assume that the graph is a t -spanner. A geometric graph is a t -spanner if for any two vertices, the length of the shortest path between them in the graph is at most t times larger than the Euclidean distance between them. Road networks, in particular in urban areas, are typically good spanners [4, 19]. For the example of Figure 1, it is clear that the road network is a t -spanner and λ -low density for low t and λ . Compared to previous work, our assumptions make the approach significantly more applicable on real-life road networks.

In this paper, we solve the *map-matching query problem* under realistic assumptions:

► **Problem 2.** *Given a geometric graph P , construct a data structure that can answer the following queries: for a polygonal curve Q in \mathbb{R}^2 ,*

1. *compute $\min_{\pi} d_{\text{F}}(\pi, Q)$ and*
2. *report $\arg \min_{\pi} d_{\text{F}}(\pi, Q)$,*

where π ranges over all paths between two vertices in P and d_{F} denotes the Fréchet distance.



■ **Figure 1** An example road network, in Barcelona. The total road length in a disk of radius r is closer to cr^2 than cr , so this road network is not c -packed. However, the number of long edges intersecting the purple disk is small, and the red path is not much longer than the blue path, so the network is λ -low density and a t -spanner for small λ and t . Map data from OpenStreetMap [20].

We present a $(1 + \varepsilon)$ -approximation under the assumptions listed above. Our approach differentiates from previous work by Gudmundsson et al. [13] in two key aspects:

- we require the graph P to be λ -low density and a t -spanner, rather than c -packed, which is a more realistic assumption for a road network [4, 8, 19];
- we solve the problem of reporting the path that minimises the Fréchet distance, which was stated as an open problem in their paper.

In order to achieve these results, we have to use different techniques, albeit at the cost of a \sqrt{n} factor replacing a polylogarithmic factor in the running time and space. Where the paper by Gudmundsson et al. [13] uses a semi-separated pair decomposition, we construct a hierarchy of small balanced separators and store appropriate associated data to guide the search for the optimal Fréchet distance. A *balanced separator* of a graph $P = (V, E)$ is a set of vertices $S \subseteq V$ that splits P into connected components of size at most $c \cdot |V|$ for constant c . Combining the changes in analysis and the capability to report a path, we get the following result. Here $n = |V| + |E|$ for a geometric graph $P = (V, E)$, and m is the number of vertices on the query polygonal curve.

► **Theorem 3.** *Suppose we are given a λ -low-density t -spanner of complexity n and a fixed $0 < \varepsilon < 1$. Let $\chi = 1/\varepsilon^2 \log 1/\varepsilon$ and let $\varphi = (\lambda/\varepsilon^3 + t^2/\varepsilon^2)^2$. In expected time $\mathcal{O}(\lambda\chi^2 n^{5/2} \log n)$ and using $\mathcal{O}(\lambda\chi^2 n^{3/2})$ space, we can construct a data structure $(1 + \varepsilon)$ -approximating Problem 2 that performs distance queries in time $\mathcal{O}(\varphi \cdot \lambda/\varepsilon \cdot m\sqrt{n} \log mn \cdot (\log^2 n + \varphi \log n + \varphi \cdot \lambda/\varepsilon))$, and answers the reporting queries for a path of length ℓ in $\mathcal{O}(\ell/\varepsilon)$ additional time.*

In a typical setting, λ and t are small constants. We need $\mathcal{O}(n^{3/2} \cdot 1/\varepsilon^4 \cdot \log^2 1/\varepsilon)$ space, resolving to $\mathcal{O}(n^{3/2})$ for fixed ε , and we support distance queries in time $\mathcal{O}(\varepsilon^{-7} m\sqrt{n} \log mn \cdot (\log^2 n + \varepsilon^{-6} \log n + \varepsilon^{-7}))$, or $\mathcal{O}(m\sqrt{n} \log mn \cdot \log^2 n)$ for fixed ε .

The rest of the paper is organised as follows. After covering some preliminaries in Section 2, we first tackle the simpler problem of finding a path in the graph that most closely follows a line segment between two vertices of the graph in terms of the Fréchet distance in Section 3. In that setting, we find a 3-approximation. The data structure we develop there is used later for obtaining a search window for when the end points of the path are not given.

In Section 4, we generalise this to an arbitrary query line segment that does not have to start or end at a graph vertex, and show how to achieve a $(1 + \varepsilon)$ -approximation. We also describe how to report a path that corresponds to a $(1 + \varepsilon)$ -approximation. Finally, in Section 5, we show how to combine the segment queries in order to handle a complete polygonal curve.

2 Preliminaries

In this paper, we work with geometric graphs, that is, graphs embedded in the plane with straight-line edges. For a graph $P = (V, E)$, we denote its complexity with $n = |V| + |E|$.

We denote the fact that a path π goes from $u \in V$ to $v \in V$ by $\pi : u \rightsquigarrow v$; and $\pi \circ \sigma$ denotes the composition (concatenation) of paths π, σ . Denote the Euclidean distance between points $x, y \in \mathbb{R}^2$ with $\|x - y\|$. We can consider the edges weighted by defining the weight of an edge $e \in E$ as $|e| = \|u - v\|$ for e connecting $u, v \in V$. We define the *graph distance* d_P as the shortest path distance along the graph between any two vertices $u, v \in V$: $d_P(u, v) = \sum_{e \in \pi} |e|$, where $\pi : u \rightsquigarrow v$ is a shortest path in P between u and v .

We assume our input graph has low density in this paper, as defined previously. We need to define two more graph properties that we use in our construction.

► **Definition 4.** A graph $P = (V, E)$ is τ -lanky [17] if for every disk of radius $r > 0$ centred at any vertex $v \in V$, there are at most τ edges of length at least r that are cut by the disk.

Here an edge is *cut* by a disk if exactly one of its endpoints is inside the disk. It is easy to see that a τ -lanky graph has bounded degree of at most τ ; and that any λ -low-density graph is also λ -lanky. Let us also formally define a t -spanner.

► **Definition 5.** A graph $P = (V, E)$ is called a t -spanner if for any two vertices $u, v \in V$, we have $d_P(u, v) \leq t \cdot \|u - v\|$.

A query is a polygonal curve in the plane, that is, a sequence of points in \mathbb{R}^2 connected with line segments. For a query curve Q , let m be the number of points in the sequence.

3 Straight Path Queries

In this section, we present a 3-approximation to the following problem, so that for the value r that we return, we have $\min_{\pi} d_F(\pi, uv) \leq r \leq 3 \cdot \min_{\pi} d_F(\pi, uv)$.

► **Problem 6.** Given a geometric graph $P = (V, E)$, construct a data structure that can answer the following queries: for a pair of vertices $u, v \in V$, compute $\min_{\pi} d_F(\pi, uv)$, where $\pi : u \rightsquigarrow v$ is a path in P .

Let $n = |V| + |E|$. To solve Problem 6 efficiently, we impose an additional constraint on P – we require in this section that P has a graph property satisfying two criteria:

1. the property is decreasing monotone, so it holds on all induced subgraphs;
2. and any graph with the property admits a small separator.

An example of such a property is planarity: any subgraph of a planar graph is planar, and the existence of small separators in planar graphs is a classical result [18]. However, not all road networks are planar, as most road networks include bridges and tunnels. Instead, we require that P is τ -lanky [17]. It is trivial to show that any subgraph of a τ -lanky graph is also τ -lanky; and Le and Than [17] show that a τ -lanky graph of complexity n admits a balanced separator of size $\mathcal{O}(\tau\sqrt{n})$ that can be found in $\mathcal{O}(\tau n)$ expected time.

In Section 4, we show how to generalise the query to arbitrary segment endpoints and how to improve the result to a $(1 + \varepsilon)$ -approximation for any fixed $\varepsilon > 0$.

Intuition. When constructing the data structure, we can use the algorithm by Alt et al. [2] in order to compute the Fréchet distance between a line segment and a path in the graph. At query time, running that algorithm would be prohibitively slow. We also want to achieve subquadratic storage, so we cannot precompute the distances for all pairs of vertices.

Broadly, the idea is to find sufficient structure in the graph to be able to find a small set of vertices so that any path in the graph passes through at least one of these vertices; we call them *transit* vertices. Then we can precompute the distances between the optimal path and the line segment when going from any vertex of the graph to one of the transit vertices. At query time, we then only need to find an optimal transit vertex. Since we are composing two paths, the computed distance is only a 3-approximation.

More specifically, a balanced separator in a graph forms a set of transit vertices. We can compute them hierarchically and store the separators and the precomputed distances in a binary tree. With some organisation, at query time, we can efficiently find all the relevant transit vertices – the ones that may separate the two query vertices on a path.

Data structure. We construct a hierarchy of separators on the graph and store it with some extra information in a binary tree. Each node in the tree represents both an induced subgraph of P , and a separator of that induced subgraph. Consider the node i corresponding to some induced subgraph $P_i = (V_i, E_i)$ of P . Conceptually, the node represents the balanced separator S_i , so the subset of vertices of P_i , splitting it into two subgraphs A_i and B_i .

The root stores S_1 and the extra information for all pairs from $V \times S_1$, so the top-level balanced separator for the entire graph. The two children of each node correspond to the subgraphs A_i and B_i . The recursion ends when the subgraphs in the leaves have constant size. In a leaf i , assign $S_i = V_i$, so compute the distances for all pairs of vertices.

For every pair of vertices $(u, s) \in V_i \times S_i$, we store $\min_{\pi} d_F(\pi, us)$, where $\pi : u \rightsquigarrow s$ in P . (Note that a path π may leave P_i .) We call all vertices in S_i *transit* vertices; and all pairs (u, s) for which we store the distances are called *transit pairs*.

In addition, for each vertex $u \in V$, we store a pointer to the tree node i so that $u \in S_i$. There is exactly one such node in the tree for every vertex: if a vertex is part of a separator, it will not be in an induced subgraph further down in the recursion, and if it is never chosen to be in a separator, then it belongs to a leaf, which is treated as S_i in its entirety.

Construction. We construct the hierarchy top-down, computing the separators on the induced subgraphs at every level using the result of Le and Than [17]. For each transit pair (p, s) in a node, we compute the appropriate Fréchet distance in the entire graph using the algorithm by Alt et al. [2], extended by Gudmundsson et al. [13, Lemma 4.3]. As we construct the separators, we also store in a table the pointer for each vertex to the correct node.

Distance query. Suppose the query is to find the minimal Fréchet distance between the segment uv and any path between u and v . Initialise $\text{opt} = \infty$. First, we use the table to find the pointers to the two nodes in the tree i and j so that $u \in S_i$ and $v \in S_j$. Then we find their lowest common ancestor, call it node a . For every node a' on the path from a to the root of the tree, perform the following procedure, updating opt . At the end, return opt .

Denote $D_{xy} = \min_{\pi} d_F(\pi, xy)$ over all $\pi : x \rightsquigarrow y$. For the query uv , denote $D'_x = \min_{r \in uv} \|r - x\|$, so the shortest distance between x and any point on uv . For all $s \in S_{a'}$, fetch the stored D_{us} and D_{sv} and compute D'_s . Then compute $D = \max(D_{us}, D_{sv}) + D'_s$ and finally assign $\text{opt} = \min(\text{opt}, D)$.

Running time analysis. For the distance queries, we take $\mathcal{O}(1)$ time to follow the pointers; $\mathcal{O}(\log n)$ time to find the lowest common ancestor; and then $\mathcal{O}(1)$ time to check the distance per transit vertex. As we check all transit vertices on the path from the lowest common ancestor to the root, we can write down the worst-case recurrence as

$$T(k) = T(2k/3) + \mathcal{O}(\tau\sqrt{k})$$

for a graph on k vertices, since the balanced separator we use subdivides the graph into two subgraphs on at most $2k/3$ vertices. For the entire graph, this resolves to $\mathcal{O}(\tau\sqrt{n})$. This term dominates the query time.

For the construction, we need $\mathcal{O}(\tau k)$ expected time to find a separator in a graph of size k . In each node, for each of $\mathcal{O}(\tau k\sqrt{k})$ transit pairs, we compute the distance in $\mathcal{O}(n \log n)$ time. Assuming we build the tree until the leaves are of constant size, we get the recurrence

$$\begin{aligned} T(k) &= T(2k/3) + T(k/3) + \mathcal{O}(\tau k + \tau k\sqrt{k} \cdot n \log n) \\ &= T(2k/3) + T(k/3) + \mathcal{O}(\tau k\sqrt{k} \cdot n \log n), \end{aligned}$$

which resolves to $\mathcal{O}(\tau n^{5/2} \log n)$ expected time overall.

Space. We store a table of pointers of size $\mathcal{O}(n)$ and the main data structure. For a graph on k vertices, we store constant-size data for each transit pair; and there are $\mathcal{O}(\tau k\sqrt{k})$ transit pairs. Overall, the space used resolves to $\mathcal{O}(\tau n\sqrt{n})$, as follows from the recurrence

$$T(k) = T(2k/3) + T(k/3) + \mathcal{O}(\tau k\sqrt{k}).$$

Correctness. It remains to show that the described query procedure gives us an appropriate distance. First, assume that we do consider an optimal transit vertex; we show that we indeed compute a 3-approximation. The following proof is essentially given by Gudmundsson et al. [13, Theorem 4.1], and relies on a semi-separated pair decomposition [10, Lemma 5.5] rather than separators. We include the proof for the sake of completeness and ease of reading.

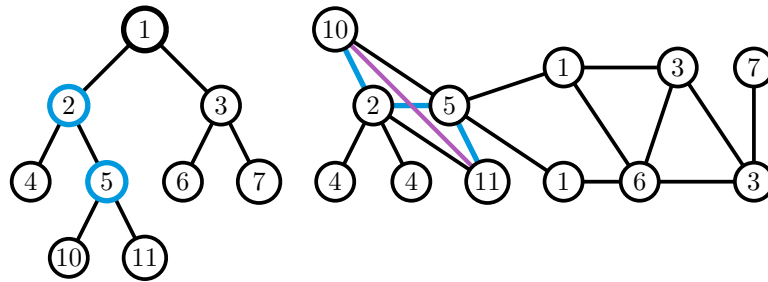
► **Lemma 7.** *Suppose that $\text{opt} = \min_{\pi'} d_{\text{F}}(\pi', uv)$ for query uv and $\pi' : u \rightsquigarrow v$, and that $\pi = \arg \min_{\pi'} d_{\text{F}}(\pi', uv)$ passes through a transit vertex s , so $\pi : u \rightsquigarrow s \rightsquigarrow v$. Let s' be the transit vertex that minimises $D = \max(D_{us'}, D_{s'v}) + D'_{s'}$. If s is considered when finding D , then $\text{opt} \leq D \leq 3 \cdot \text{opt}$.*

Proof. Let t be the point on uv closest to s' . Let $\pi_{us'} = \arg \min_{\pi'} d_{\text{F}}(\pi', us')$ over $\pi' : u \rightsquigarrow s'$, and define $\pi_{s'v}$ similarly. Note that the composition of these paths $\pi_{us'} \circ \pi_{s'v}$ does not have to be the same as π . Then

$$\begin{aligned} \text{opt} = d_{\text{F}}(\pi, uv) &\leq d_{\text{F}}(\pi_{us'} \circ \pi_{s'v}, uv) \leq \max(d_{\text{F}}(\pi_{us'}, ut), d_{\text{F}}(\pi_{s'v}, tv)) \\ &\leq \max(d_{\text{F}}(\pi_{us'}, us') + \|s' - t\|, d_{\text{F}}(\pi_{s'v}, s'v) + \|s' - t\|) \\ &= \|s' - t\| + \max(d_{\text{F}}(\pi_{us'}, us'), d_{\text{F}}(\pi_{s'v}, s'v)) \\ &= D'_{s'} + \max(D_{us'}, D_{s'v}) = D. \end{aligned}$$

On the other hand, note that $D \leq \max(D_{us}, D_{sv}) + D'_s$, as s' minimises that expression. We also have $D'_s \leq d_{\text{F}}(\pi, uv)$. Let $\pi(u, s)$ be the subpath of π from u to s . Let r be the point in uv that is aligned to s in the Fréchet alignment between π and uv . Then

$$\begin{aligned} D_{us} = d_{\text{F}}(\pi_{us}, us) &\leq d_{\text{F}}(\pi(u, s), us) \leq d_{\text{F}}(\pi(u, s), ur) + d_{\text{F}}(ur, us) \\ &= d_{\text{F}}(\pi(u, s), ur) + \|r - s\| \\ &\leq d_{\text{F}}(\pi, uv) + d_{\text{F}}(\pi, uv) = 2d_{\text{F}}(\pi, uv). \end{aligned}$$



■ **Figure 2** A representation of the hierarchy (left) for the graph (right). A query segment is shown in purple, a possible path in blue. We check nodes 5, 2, and 1. If we pick the transit vertex in 5, then the path may be $10 \rightarrow 2 \rightarrow 5$, so we may need to go up the tree to find the next transit pair.

Using the same argument for D_{sv} , we conclude

$$D \leq D'_s + \max(D_{us}, D_{sv}) \leq d_F(\pi, uv) + 2d_F(\pi, uv) = 3d_F(\pi, uv),$$

and so $\text{opt} \leq D \leq 3 \cdot \text{opt}$ and the value is a 3-approximation. ◀

Now we show that we consider all the relevant transit vertices. See Figure 2.

► **Lemma 8.** *For the query uv , the procedure considers a transit vertex s such that s lies on an optimal path $\pi = \arg \min_{\pi'} d_F(\pi', uv)$, where $\pi' : u \rightsquigarrow v$.*

Proof. We consider two cases based on where the lowest common ancestor is found. First, suppose that the lowest common ancestor a contains u , v , or both u and v in the separator. In other words, $u \in S_a$ or $v \in S_a$, and so $s = u$ or $s = v$. Then π passes through s , and we consider s as a transit vertex.

Now assume that the lowest common ancestor a does not contain u or v in the separator; then u and v are separated by S_a . Without loss of generality, let $u \in A_a$ and $v \in B_a$. If the path π stays within the subgraph P_a , then it goes through some $s \in S_a$, which we consider. Otherwise, it goes through some separator between P_a and the rest of the graph; we check exactly all the vertices in these separators, as they fall on the path from a to the root. ◀

Bringing the above considerations together, we get the main result of this section.

► **Theorem 9.** *Given a τ -lanky graph of complexity n , we can construct a data structure giving a 3-approximation for Problem 6 in expected time $\mathcal{O}(\tau n^{5/2} \log n)$, using $\mathcal{O}(\tau n \sqrt{n})$ space, that supports distance queries in time $\mathcal{O}(\tau \sqrt{n})$.*

4 Map-Matching Segment Queries

In this section, we generalise the construction we just presented to compute a $(1 + \varepsilon)$ -approximation and to handle reporting, as well as to support arbitrary query line segments.

► **Problem 10.** *Given a geometric graph $P = (V, E)$, construct a data structure that can answer the following queries: for a line segment pq in the plane,*

1. *compute $\min_{\pi} d_F(\pi, pq)$ and*
2. *report $\arg \min_{\pi} d_F(\pi, pq)$,*

where π ranges over all paths between two vertices in P .

For distance queries, we closely follow the work of Gudmundsson et al. [13], which in turn follows the approach of Driemel and Har-Peled [10]. The latter appears at first to be devoted to a rather different problem, but turns out to be very helpful in the map-matching setting.

Data structure for distance queries with fixed path endpoints. We can immediately use the approach of Section 3 on arbitrary segments. Suppose the query is a pair of vertices $u, v \in V$ and a segment $pq \subset \mathbb{R}^2$. Then we can find a 3-approximation to $\min_{\pi} d_{\mathbb{F}}(\pi, pq)$, where $\pi : u \rightsquigarrow v$. To achieve that, we just need to define $D'_{s'} = d_{\mathbb{F}}(us' \circ s'v, pq)$ and let t be the point on pq aligned with s' under this matching.

We can directly use the following statement [13, Lemma 5.2], which closely mimics the approach of Driemel and Har-Peled [10, Lemma 5.8]:

► **Lemma 11.** *Let $u, v \in V$ be a fixed pair of vertices. Let $\varepsilon > 0$ and $\chi = 1/\varepsilon^2 \log 1/\varepsilon$. In $\mathcal{O}(\chi^2 n \log n)$ time and using $\mathcal{O}(\chi^2)$ space, one can construct a data structure that, given a query segment pq in the plane, returns in $\mathcal{O}(1)$ time a $(1+\varepsilon)$ -approximation to $\min_{\pi} d_{\mathbb{F}}(\pi, pq)$, where $\pi : u \rightsquigarrow v$.*

The idea behind this lemma is to construct an exponential grid around both fixed vertices so that the grid is denser close to the vertices. There is an upper bound and a lower bound on how far the grid goes, which is based on ε and $\min_{\pi} d_{\mathbb{F}}(\pi, uv)$. If the segment pq is closer to uv than the smallest grid cell, then taking $\min_{\pi} d_{\mathbb{F}}(\pi, uv)$ gives us a good approximation; if the segment is very far, then $d_{\mathbb{F}}(pq, uv)$ dominates. Otherwise, we are guaranteed that there are grid points p' and q' that match p and q closely with respect to u and v . We can simply precompute $\min_{\pi} d_{\mathbb{F}}(\pi, p'q')$ for all pairs of points p' and q' and return an appropriate value in constant time when given a query. See Figure 3.

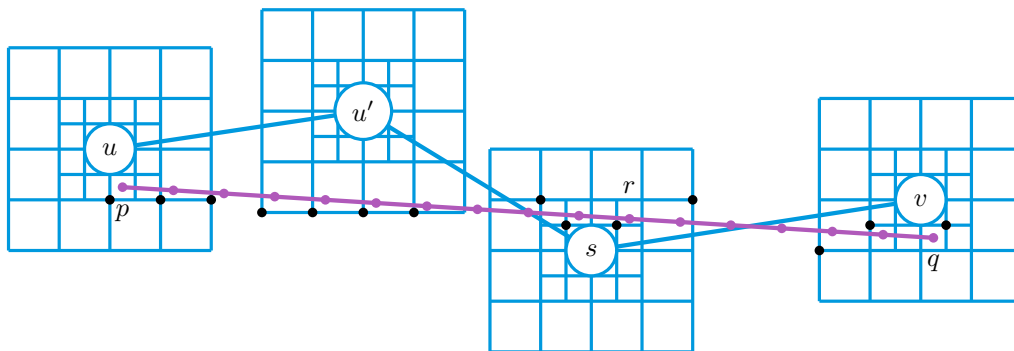
In order to improve the approximation ratio to $1+\varepsilon$, we use Lemma 5.3 by Gudmundsson et al. [13], substituting our data structure of Theorem 9 for their data structure of Lemma 5.1. The argument is the same: we can construct a grid around each graph vertex and precompute the distances for all pairs of grid vertices for each transit pair; and we can store that in the data structure of Theorem 9. At query time, when testing each transit vertex s , we subsample the relevant part of segment pq with $\mathcal{O}(1/\varepsilon)$ points to find an optimal spot that should align with s . We use the data structure of Theorem 9 to make sure we do not need to sample too many points. With our time and space bounds, we get the following lemma.

► **Lemma 12.** *Let $\varepsilon > 0$ and $\chi = 1/\varepsilon^2 \log 1/\varepsilon$. In expected time $\mathcal{O}(\tau \chi^2 n^{5/2} \log n)$ and using $\mathcal{O}(\tau \chi^2 n \sqrt{n})$ space, we can construct a data structure that, given a query segment pq in the plane and a pair of vertices $u, v \in V$, returns in $\mathcal{O}(\tau/\varepsilon \sqrt{n})$ time a $(1+\varepsilon)$ -approximation to $\min_{\pi} d_{\mathbb{F}}(\pi, pq)$, where $\pi : u \rightsquigarrow v$.*

Reporting a path. Next we discuss the modifications needed to report a curve that realises the $(1+\varepsilon)$ -approximate distance. We can perform an approximate distance query first. Once we have the distance, we can find the transit pairs that realise it; with these pairs, we can store the next vertex on the optimal path. We can then repeat these queries with the new pairs. It remains to show how to perform this sequence of queries consistently, i.e. so that an approximate route for a subpath also approximates the complete path.

Recall that when computing the $(1+\varepsilon)$ -approximation, we consider a ball of a certain radius around a transit vertex, and we take $\mathcal{O}(1/\varepsilon)$ sample points on the query segment pq inside the ball, to test the Fréchet alignment with the transit vertex. The first modification is that we impose fixed coordinates for the sample points. For some fixed point on the line pq and for some constant c , every sample point is $\mathcal{O}(c/\varepsilon)$ distance away from the fixed point.

Next, we describe the necessary modifications to the data structure of Lemma 12. With each transit pair and for each pair of grid points, in addition to the Fréchet distance, we also store the first vertex on the optimal path, so $u' \in V$ such that for $\pi' = \arg \min_{\pi} d_{\mathbb{F}}(\pi, pr)$, we have $\pi' : u \rightarrow u' \rightsquigarrow s$. (Here r is the point on pq that maps to s .)



■ **Figure 3** A query trajectory pq is shown in purple, and the reported path in the graph is shown in blue. We sample points on pq at regular distance and snap them to the exponential grid around the graph vertices. Once we find r on pq that aligns with the transit vertex s , we can query the pair (u, s) with the (snapped) segment pr to find the next vertex u' .

The query proceeds as follows. First, we perform the distance query for pq and record the optimal transit vertex s . Find the point r among the $\mathcal{O}(1/\varepsilon)$ samples on pq that aligns with s . Query the pairs (u, s) and (s, v) with pr and rq , respectively, and retrieve the stored adjacent vertices u' and v' . Again, find the optimal alignment points on pr and rq ; find pairs (u'', s) and (s, v'') ; repeat until the complete path is reported. In the special case when $s = v$ or $s = u$, only one sequence of queries has to be performed. If u and v are both in a leaf, we can proceed as if $s = v$. See Figure 3.

Suppose the transit vertex s is stored in some S_i . If the optimal path leaves P_i , then it is possible that u' is not in P_i , and so the pair (u', s) is not stored in node i . However, then u' must be in some separator separating P_i from a different subgraph P_j . Furthermore, note that the separator in question must be on the path from i to the root. Thus, we can go up until we find $u' \in S_j$ for some $j < i$. We can continue the procedure, now for the transit pair (s, u') , finding some s' so that the path is of the shape $s \rightarrow s' \rightsquigarrow u'$. See Figures 2 and 3.

We briefly analyse the time and space bounds. We find the next vertex on the path from the free-space diagram when computing the Fréchet distance. As we store constant extra information, the preprocessing and space bounds are unchanged. For the query time, in addition to the distance query, we report a path of length ℓ . To find each next vertex, we find the correct transit pair in constant time, then test $\mathcal{O}(1/\varepsilon)$ alignment options. We may have to go up the tree; however, as we never go down the tree, that traversal happens only once per query. Therefore, the extra time needed to report the path with ℓ vertices is $\mathcal{O}(\log n + \ell/\varepsilon)$. It remains to show that the reported path indeed corresponds to a $(1 + \varepsilon)$ -approximation.

► **Lemma 13.** *For query pq , if $\pi = \arg \min_{\pi^*} d_F(\pi^*, pq)$ has the shape $\pi : u \rightarrow u' \rightsquigarrow s$, and u' is aligned to some $p' \in pq$ under the Fréchet alignment, then $\pi' = \arg \min_{\pi^*} d_F(\pi^*, p'q)$ of the shape $\pi' : u' \rightsquigarrow s$ is a subpath of π .*

Proof. Without loss of generality, we can assume that s is a transit vertex. If the distances were computed exactly, the statement would clearly hold. We need to show that the sampling and the grid do not introduce inconsistencies.

Recall that the sample points are placed on the line segment independently of context. Therefore, the location of sample points is the same on pq and $p'q$. Furthermore, we always snap these original sample points to the grid, and the grid does not depend on the path. Therefore, we can view pq as a sequence of grid points that all possible sample points would snap to; and $p'q$ then snaps to a subsequence of those grid points. For the pairs

27:10 Map-Matching Queries Under Fréchet Distance on Low-Density Spanners

of grid points, the distances are computed directly. Therefore, we do not introduce any additional error, compared to a distance query, and so the reported path corresponds to a $(1 + \varepsilon)$ -approximation. ◀

Data structure for finding the path endpoints. So far, we only required that P is τ -lanky. For the next data structure, we also need P to be a t -spanner. Recall that $d_P(u, v)$ denotes the shortest path distance in the graph between the vertices u and v . If $P = (V, E)$ is a t -spanner, then for any $u, v \in V$, we have $d_P(u, v) \leq t \cdot \|u - v\|$. To solve Problem 10, we need one more data structure. Lemma 12 still requires us to pick vertices u and v to check the paths $\pi : u \rightsquigarrow v$. We need to be able to select a subset of candidate vertices so that we can obtain a $(1 + \varepsilon)$ -approximation, but the subset is still small enough. To that aim, we perform the same procedure as Gudmundsson et al. [13], but get different bounds.

In particular, we run Gonzalez's k -centre clustering algorithm [12] for $k = n$ on the vertices of the graph $P = (V, E)$ using the distance d_P . In short, the algorithm selects cluster centres from V iteratively, starting with a random one; and each following one is the furthest away from any other centre. Let c_1 be the first (random) centre. Define the *radius* of a clustering to be the maximum distance from any vertex to its closest centre. Denote $C_i = \{c_1, \dots, c_i\}$. Then for all $2 \leq i \leq n$, we compute

$$c_i = \arg \max_{v \in V} \min_{c \in C_{i-1}} d_P(v, c), \quad r_i = \max_{v \in V} \min_{c \in C_i} d_P(v, c).$$

We obtain a sequence $\langle (C_1, r_1), \dots, (C_n, r_n) \rangle$, where $r_n = 0$, since all vertices are centres. Using this sequence, we can show the following lemma.

► **Lemma 14.** *Let $P = (V, E)$ be a t -spanner and let S be a square in the plane with side length $2r$. Then there exists a set of vertices $T \subseteq V$ satisfying two properties:*

1. $|T| = \mathcal{O}((t/\varepsilon)^2)$, and
2. for all $v \in V \cap S$, there is $z \in T$ such that $d_P(v, z) \leq \varepsilon r$.

Proof. If $r_1 < \varepsilon r$, pick $T = \{c_1\}$; then the first property holds immediately, and the second property holds by definition of r_1 . Otherwise, note that $r_n = 0$, so now let i be the index so that $r_i \geq \varepsilon r$ and $r_{i+1} < \varepsilon r$. Take S' to be the square concentric with S , but with the side length of $4r$, and let $T = C_{i+1} \cap S'$. The second property is immediately satisfied, since any vertex, including those in S , is closer than εr to some centre; and choosing S' this way ensures that we cannot exclude any relevant centres, since $\varepsilon < 1$.

To see that the first property is true, consider C_i . Due to the sequence of picking centres, we know that any two vertices c_j and c_ℓ with $j < \ell$ in T are far apart, i.e. $d_P(c_j, c_\ell) \geq \varepsilon r$. If this were not true, then c_ℓ would not have been chosen as a centre, since there still are vertices in V that are further than εr away from any centre. But then we know

$$\varepsilon r \leq d_P(c_j, c_\ell) \leq t \cdot \|c_j - c_\ell\|,$$

so any two points are at least $\varepsilon/t \cdot r$ apart in the plane. In a square with side length $4r$, we can only pack $\mathcal{O}((t/\varepsilon)^2)$ of these vertices. C_{i+1} has only one more vertex, so the first property holds for T , as well. ◀

Gudmundsson et al. [13] show how to construct a data structure based on their version of Lemma 14. The proof is the same; only the bounds change.

► **Lemma 15.** *Let $P = (V, E)$ be a t -spanner, and let $0 \leq \varepsilon \leq 1$. In $\mathcal{O}(n^2 \log n)$ time and using $\mathcal{O}(n \log n)$ space, we can construct a data structure that, given a query square S in the plane with side length $2r$, returns a set of vertices T satisfying Lemma 14 in time $\mathcal{O}(\log n + (t/\varepsilon)^2)$.*

The main theorem of this section follows using our data structures.

► **Theorem 16.** *Given a τ -lanky t -spanner of complexity n , we can construct the data structure for Problem 10 in expected time $\mathcal{O}(\tau\varepsilon^{-4} \log^2(1/\varepsilon)n^{3/2} \log n)$ and using $\mathcal{O}(\tau\varepsilon^{-4} \log^2(1/\varepsilon)n\sqrt{n})$ space, so the distance queries can be answered in time $\mathcal{O}(\tau t^8 \varepsilon^{-9} \sqrt{n} \log n (\log n + \tau/\varepsilon))$; and the reporting queries for a path of length ℓ can be answered in $\mathcal{O}(\ell/\varepsilon)$ additional time.*

Proof. The changes we made for reporting do not affect Lemma 15, so the proof of Gudmundsson et al. [13] applies. We only discuss the time bounds. Preprocessing simply consists of building the data structures for Lemmas 12 and 15. For the query time, consider first the decision version of the algorithm. We query the data structure of Lemma 15 twice; and then for every pair of possible matching vertices, we query the data structure of Lemma 12. The second step dominates, taking $\mathcal{O}(\tau t^4 \varepsilon^{-5} \sqrt{n})$ time.

For the optimisation version, we use parametric search with $N_P = \sqrt{n}$ parallel processors. The sequential version runs in the same time as the decision version, so $T_S = \mathcal{O}(\tau t^4 \varepsilon^{-5} \sqrt{n})$. In the parallel version, querying the distance data structure can be done with \sqrt{n} processors, each performing $\mathcal{O}(\tau/\varepsilon)$ amount of work, then combining the values to find the minimum in $\mathcal{O}(\log n)$ time. Thus, $T_P = \mathcal{O}((t/\varepsilon)^4 \cdot (\tau/\varepsilon + \log n))$. The time for the optimisation version is now $\mathcal{O}(N_P T_P + T_P T_S \log N_P)$. This amounts to $\mathcal{O}(\tau t^8 \varepsilon^{-9} \sqrt{n} \log n (\log n + \tau/\varepsilon))$.

For the reporting query, perform the distance query and record the optimal path endpoints; then, as we discussed, it costs extra $\mathcal{O}(\ell/\varepsilon)$ time to report a path of length ℓ . ◀

5 General Map-Matching Queries

In this section, we generalise the problem again to handle a polygonal curve rather than a line segment as a query. The procedure is very similar; however, we want to make sure that the Fréchet alignment between a query curve and a path can align vertices of the query to points on graph edges, and not just to graph vertices. To that effect, we need to extend Lemma 14 so we can sample a small number of points on graph edges.

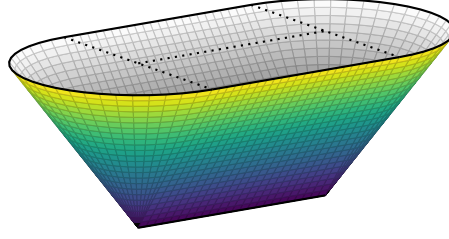
Gudmundsson et al. [13] use c -packedness again; however, in our setting, the t -spanner property is not sufficient, as it does not give us guarantees about the graph distance between points on the edges. Here we require the graph to also be λ -low density.

► **Lemma 17.** *Let $P = (V, E)$ be a λ -low-density t -spanner, let $F = \{f \in \mathbb{R}^2 \mid f \in e, e \in E\}$, and let S be a square in the plane with side length $2r$. Then there exists a set of points $T \subseteq F$ satisfying two properties:*

1. $|T| = \mathcal{O}(t^2/\varepsilon^2 + \lambda/\varepsilon^3)$, and
2. for all $p \in F \cap S$, there is $z \in T$ such that $d_P(p, z) \leq \varepsilon r$.

Proof. We can use Lemma 14 with $\varepsilon' = \varepsilon/2$ to obtain the set T_1 of size $\mathcal{O}((t/\varepsilon')^2)$ so that for all $v \in V \cap S$, $d_P(p, z) \leq \varepsilon' r$ for some $z \in T_1$. Define $E_r \subseteq E$ to contain the edges of length at least εr . Let S' be a square concentric with S but with the side length $4r$. For each $e \in E_r$, choose $\mathcal{O}(1/\varepsilon)$ evenly spaced points on $e \cap S'$ with the distance between them of at most εr . Let T_2 be the set of all such points, and assign $T = T_1 \cup T_2$.

We first show that the first property holds. For T_2 , we need to bound the size of $E_r \cap S'$. As P is λ -low density, we know that there are at most λ edges of length at least εr intersecting any disk of diameter εr , and every edge has $\mathcal{O}(1/\varepsilon)$ sample points. We can cover S' with $\mathcal{O}((1/\varepsilon)^2)$ such disks, so $|T_2| = \mathcal{O}(\lambda(1/\varepsilon)^3)$. Therefore, $|T| = \mathcal{O}(t^2/\varepsilon^2 + \lambda/\varepsilon^3)$.



■ **Figure 4** The surface of a trough: at fixed z , we include all points no further than $4z$ from e .

Now consider the second property. Note that $V \subset F$. For any $v \in V \cap S$, we immediately conclude that the property holds by Lemma 14. For any $p \in e \cap S$, $e \in E$ with $|e| \leq \varepsilon r$, note that both endpoints of e lie in S' . So there is a vertex $v \in V \cap S'$ so that $d_P(p, v) \leq \varepsilon' r$, and by Lemma 14, $d_P(v, z) \leq \varepsilon' r$ for some $z \in T_1$. Therefore, $d_P(p, z) \leq 2\varepsilon' r = \varepsilon r$. Finally, for any $p \in e \cap S$, $e \in E_r$, it is clear that there is a point not further than εr in T_2 . ◀

We will build a data structure similar to Gudmundsson et al.'s [13]. We start by stating a definition of λ -low density in \mathbb{R}^3 .

► **Definition 18.** *A set of objects in \mathbb{R}^3 is k -low density if, for every axis-parallel cube H_r with side length r , there are at most k objects of size at least r that intersect H_r . The size of an object is the side length of its smallest axis-parallel enclosing cube.*

► **Definition 19.** *Given a segment $e \subset \mathbb{R}^2$ and $0 < \varepsilon < 1$, define*

$$\text{trough}(e, \varepsilon) = \{(x, y, z) \in \mathbb{R}^3 \mid d((x, y), e) \leq 4z \leq 8|e|/\varepsilon\},$$

where $d((x, y), e)$ is the distance from (x, y) to the closest point on e . See Figure 4.

A trough is a three-dimensional object consisting of a triangular prism and two half-cones. Any z -slice can be seen as a z -neighbourhood of e . We show the following lemma.

► **Lemma 20.** *Let $P = (V, E)$ be λ -low density, and let $0 < \varepsilon < 1$. The set $\{\text{trough}(e, \varepsilon) \mid e \in E\}$ is k -low density for $k = \mathcal{O}(\lambda/\varepsilon^2)$.*

Proof. We first bound the size of $\text{trough}(e, \varepsilon)$. Let $(x, y, z) \in \text{trough}(e, \varepsilon)$. Note that $0 \leq z \leq 2|e|/\varepsilon$. Furthermore, $d((x, y), e) \leq 8|e|/\varepsilon$, so (x, y) must lie inside a disk centred at the midpoint of e with radius $9|e|/\varepsilon$. Thus (x, y, z) lies inside a cube with side length $18|e|/\varepsilon$, which bounds the size of $\text{trough}(e, \varepsilon)$.

Let H_r be an axis-parallel cube with side length r , and let its smallest z -coordinate be $z_{\min} \geq 0$. Suppose $\text{trough}(e, \varepsilon)$ of size at least r intersects H_r , and let (x, y, z) be a point in the intersection. Let h be the projection of the centre of H_r onto the plane $z = 0$. Then

$$d(h, e) \leq d(h, (x, y)) + d((x, y), e) \leq r + 4z \leq 5r + 4z_{\min},$$

where the first step follows by the triangle inequality, the second by (x, y, z) lying in the intersection, and the third by $z \leq z_{\min} + r$. Furthermore, the size of $\text{trough}(e, \varepsilon)$ is at least r and at most $18|e|/\varepsilon$, so $r \leq 18|e|/\varepsilon$; and $z_{\min} \leq 2|e|/\varepsilon$. Therefore, $5r + 4z_{\min} \leq 98|e|/\varepsilon$.

So we know $|e| \geq (5r + 4z_{\min}) \cdot \varepsilon/98$. By λ -low-density property, any ball with diameter $(5r + 4z_{\min}) \cdot \varepsilon/98$ is intersected by at most λ such edges. Consider a disk in $z = 0$ with diameter $2 \cdot (5r + 4z_{\min})$ centred at h . It can be covered by c/ε^2 smaller disks for a constant c , so there may be at most $k = c\lambda/\varepsilon^2$ edges close enough to h for H_r to intersect their troughs; and so the set of troughs is k -low density. ◀

Using the range searching data structure for low-density sets by Schwarzkopf and Vleugels [21] on the set of troughs of all edges, we obtain the following result.

► **Lemma 21.** *Let $P = (V, E)$ be a λ -low-density t -spanner, let $0 \leq \varepsilon \leq 1$, and let $F = \{f \in \mathbb{R}^2 \mid f \in e, e \in E\}$. In $\mathcal{O}(n^2 \log n + \lambda/\varepsilon^2 \cdot n \log n)$ time and using $\mathcal{O}(n \log^2 n + n \cdot \lambda/\varepsilon^2)$ space, we can construct a data structure that, given a query square S in the plane with side length $2r$, returns a set of vertices T satisfying Lemma 17 in time $\mathcal{O}(\log^2 n + t^2/\varepsilon^2 + \lambda/\varepsilon^3)$.*

Finally, we obtain the main result of the paper. The proof of Gudmundsson et al. [13, Theorem 6.1] applies here directly, but we use the data structures in Lemmas 12 and 21.

► **Theorem 3.** *Suppose we are given a λ -low-density t -spanner of complexity n and a fixed $0 < \varepsilon < 1$. Let $\chi = 1/\varepsilon^2 \log 1/\varepsilon$ and let $\varphi = (\lambda/\varepsilon^3 + t^2/\varepsilon^2)^2$. In expected time $\mathcal{O}(\lambda\chi^2 n^{5/2} \log n)$ and using $\mathcal{O}(\lambda\chi^2 n^{3/2})$ space, we can construct a data structure $(1+\varepsilon)$ -approximating Problem 2 that performs distance queries in time $\mathcal{O}(\varphi \cdot \lambda/\varepsilon \cdot m\sqrt{n} \log mn \cdot (\log^2 n + \varphi \log n + \varphi \cdot \lambda/\varepsilon))$, and answers the reporting queries for a path of length ℓ in $\mathcal{O}(\ell/\varepsilon)$ additional time.*

Proof. See the proof by Gudmundsson et al. [13, Theorem 6.1], but we use Lemmas 12 and 21 instead. We analyse the space and time requirements. For preprocessing and space, we construct the two data structures. For distance queries, first analyse the decision version.

We query the data structure of Lemma 21 m times to obtain the candidate points. Then we construct a directed graph with $\mathcal{O}(m \cdot (\lambda/\varepsilon^3 + t^2/\varepsilon^2)^2)$ edges. For each edge, we do a constant number of queries to the data structure of Lemma 12, each taking $\mathcal{O}(\sqrt{n} \cdot \lambda/\varepsilon)$ time. Finally, we decide if there is a suitable directed path in the graph. Overall, the decision version takes $\mathcal{O}(m\sqrt{n} \cdot \lambda/\varepsilon \cdot (\lambda/\varepsilon^3 + t^2/\varepsilon^2)^2)$ time.

For the optimisation version, we apply parametric search using $N_P = m\sqrt{n}$ parallel processors. The sequential version runs in the same time as the decision version, so $T_S = \mathcal{O}(m\sqrt{n} \cdot \lambda/\varepsilon \cdot (\lambda/\varepsilon^3 + t^2/\varepsilon^2)^2)$. In the parallel version, the steps for each of m points can be executed in parallel; and finding the weight of an edge by querying the distance data structure can be done with \sqrt{n} processors, each performing $\mathcal{O}(\lambda/\varepsilon)$ amount of work, then combining the values to find the minimum in $\mathcal{O}(\log n)$ time. Thus, $T_P = \mathcal{O}(\log^2 n + (\lambda/\varepsilon + \log n) \cdot (\lambda/\varepsilon^3 + t^2/\varepsilon^2)^2)$. The time for the optimisation version is now $\mathcal{O}(N_P T_P + T_P T_S \log N_P)$. Let $\varphi = (\lambda/\varepsilon^3 + t^2/\varepsilon^2)^2$; then the query time is

$$\mathcal{O}(\varphi \cdot \lambda/\varepsilon \cdot m\sqrt{n} \log mn \cdot (\log^2 n + \varphi \log n + \varphi \cdot \lambda/\varepsilon)).$$

Treating t and λ as constant, we get $\mathcal{O}(\varepsilon^{-7} \cdot m\sqrt{n} \log mn \cdot (\log^2 n + \varepsilon^{-6} \log n + \varepsilon^{-7}))$; and treating also ε as a constant, the query time becomes $\mathcal{O}(m\sqrt{n} \log mn \cdot \log^2 n)$.

Finally, for the reporting query, we first run the distance query and record the path in the graph, as well as the edges aligned with the query curve vertices, and record the optimal order of endpoints of these edges. Now we can simply perform the individual segment reporting queries, as before, costing us extra $\mathcal{O}(\ell/\varepsilon)$ time for a path of length ℓ . ◀

References

- 1 Mohamed Ali, John Krumm, Travis Rautman, and Ankur Teredesai. ACM SIGSPATIAL GIS cup 2012. In Isabel Cruz, Craig Knoblock, Peer Kröger, Egemen Tanin, and Peter Widmayer, editors, *Proceedings of the 20th International Conference on Advances in Geographic Information Systems (SIGSPATIAL 2012)*, pages 597–600, New York, NY, USA, 2012. ACM. doi:10.1145/2424321.2424426.
- 2 Helmut Alt, Alon Efrat, Günter Rote, and Carola Wenk. Matching planar maps. *Journal of Algorithms*, 49(2):262–283, 2003. doi:10.1016/S0196-6774(03)00085-3.
- 3 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5(1):75–91, 1995. doi:10.1142/S0218195995000064.
- 4 Boris Aronov, Kevin Buchin, Maike Buchin, Bart Jansen, Tom de Jong, Marc van Kreveld, Maarten Löffler, Jun Luo, Rodrigo I. Silveira, and Bettina Speckmann. Connect the dot: Computing feed-links for network extension. *Journal of Spatial Information Science*, 3:3–31, 2011. doi:10.5311/JOSIS.2011.3.47.
- 5 Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk. On map-matching vehicle tracking data. In Kjell Bratbergsengen, editor, *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB 2005)*, pages 853–864, Los Angeles, CA, USA, 2005. VLDB Endowment. URL: <https://dl.acm.org/doi/10.5555/1083592.1083691>.
- 6 Erin 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. doi:10.1145/3368617.
- 7 Pingfu Chao, Yehong Xu, Wen Hua, and Xiaofang Zhou. A survey on map-matching algorithms. In Renata Borovica-Gajic, Jianzhong Qi, and Weiqing Wang, editors, *Databases Theory and Applications: 31st Australasian Database Conference (ADC 2020)*, volume 12008 of *Lecture Notes in Computer Science*, pages 121–133, Berlin, Germany, 2020. Springer. doi:10.1007/978-3-030-39469-1_10.
- 8 Daniel Chen, Anne Driemel, Leonidas J. Guibas, Andy Nguyen, and Carola Wenk. Approximate map matching with respect to the Fréchet distance. In Matthias Müller-Hannemann and Renato Werneck, editors, *Proceedings of the 2011 Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 75–83, Philadelphia, PA, USA, 2011. SIAM. doi:10.1137/1.9781611972917.8.
- 9 Daniel Chen, Christian Sommer, and Daniel Wolleb. Fast map matching with vertex-monotone Fréchet distance. In Matthias Müller-Hannemann and Federico Perea, editors, *21st Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2021)*, volume 96 of *Open Access Series in Informatics (OASISs)*, pages 10:1–10:20, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASISs.ATMOS.2021.10.
- 10 Anne Driemel and Sarel Har-Peled. Jaywalking your dog: Computing the Fréchet distance with shortcuts. *SIAM Journal on Computing*, 42(5):1830–1866, 2013. doi:10.1137/120865112.
- 11 Michael Godau. A natural metric for curves: Computing the distance for polygonal chains and approximation algorithms. In Christian Choffrut and Matthias Jantzen, editors, *Proceedings of the 8th Annual Symposium on Theoretical Aspects of Computer Science (STACS 1991)*, volume 480 of *Lecture Notes in Computer Science*, pages 127–136, Berlin, Germany, 1991. Springer. doi:10.1007/BFb0020793.
- 12 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985. doi:10.1016/0304-3975(85)90224-5.
- 13 Joachim Gudmundsson, Martin P. Seybold, and Sampson Wong. Map matching queries on realistic input graphs under the Fréchet distance. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 Annual ACM–SIAM Symposium on Discrete Algorithms (SODA)*, pages 1464–1492, Philadelphia, PA, USA, 2023. SIAM. doi:10.1137/1.9781611977554.ch53.

- 14 Joachim Gudmundsson and Michiel Smid. Fast algorithms for approximate Fréchet matching queries in geometric trees. *Computational Geometry: Theory & Applications*, 48(6):479–494, 2015. doi:10.1016/j.comgeo.2015.02.003.
- 15 Mahdi Hashemi and Hassan A. Karimi. A critical review of real-time map-matching algorithms: Current issues and future directions. *Computers, Environment and Urban Systems*, 48:153–165, 2014. doi:10.1016/j.compenvurbsys.2014.07.009.
- 16 Matej Kubicka, Arben Cela, Hugues Mounier, and Silviu-Iulian Niculescu. Comparative study and application-oriented classification of vehicular map-matching methods. *IEEE Intelligent Transportation Systems Magazine*, 10(2):150–166, 2018. doi:10.1109/MITS.2018.2806630.
- 17 Hung Le and Cuong Than. Greedy spanners in Euclidean spaces admit sublinear separators. In Joseph Naor and Niv Buchbinder, editors, *Proceedings of the 2022 Annual ACM–SIAM Symposium on Discrete Algorithms (SODA)*, pages 3287–3310, Philadelphia, PA, USA, 2022. SIAM. doi:10.1137/1.9781611977073.130.
- 18 Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979. doi:10.1137/0136016.
- 19 Stig Nordbeck. Computing distances in road networks. *Papers in Regional Science*, 12(1):207–220, 1964. doi:10.1111/j.1435-5597.1964.tb01266.x.
- 20 OpenStreetMap. Map data, 2023. URL: <https://openstreetmap.org>.
- 21 Otfried Schwarzkopf and Jules Vleugels. Range searching in low-density environments. *Information Processing Letters*, 60(3):121–127, 1996. doi:10.1016/S0020-0190(96)00154-8.
- 22 Martin P. Seybold. Robust map matching for heterogeneous data via dominance decompositions. In Nitesh Chawla and Wei Wang, editors, *Proceedings of the 2017 SIAM International Conference on Data Mining (SDM)*, pages 813–821, Philadelphia, PA, USA, 2017. SIAM. doi:10.1137/1.9781611974973.91.
- 23 A. Frank van der Stappen. *Motion Planning Amidst Fat Obstacles*. PhD thesis, Universiteit Utrecht, 1994. URL: https://webpace.science.uu.nl/~stapp101/PhDThesis_AFvanderStappen.pdf.
- 24 Hong Wei, Yin Wang, George Forman, and Yanmin Zhu. Map matching: Comparison of approaches using sparse and noisy data. In Craig Knoblock, Peer Kröger, John Krumm, Markus Schneider, and Peter Widmayer, editors, *Proceedings of the 21st International Conference on Advances in Geographic Information Systems (SIGSPATIAL 2013)*, pages 444–447, New York, NY, USA, 2013. ACM. doi:10.1145/2525314.2525456.
- 25 Yu Zheng and Xiaofang Zhou, editors. *Computing with Spatial Trajectories*. Springer, Berlin, Germany, 2011. doi:10.1007/978-1-4614-1629-6.