



A Practical Guide to an Open-Source Map-Matching Approach for Big GPS Data

Siavash Saki¹ · Tobias Hagen¹

Received: 11 March 2022 / Accepted: 19 July 2022 / Published online: 4 August 2022
© The Author(s) 2022

Abstract

This work shows how map-matching helps to minimize errors in GPS data by finding the most probable corresponding points of the recorded waypoints of a trajectory on a road network. We investigate an open-source alternative for map-matching trajectories called Valhalla, which could replace limited and costly commercial map-matching services. Valhalla is an open-source routing engine, which provides different services, such as path-finding, map-matching, and generating maneuvers based on a path. We build a cloud-based big data analytics framework on Amazon Web Services (AWS) platform for map-matching. This well-established framework is scalable and could process millions of trajectories. Using an example GPS dataset, it is demonstrated how Valhalla can be used for map-matching at scale. The dataset consists of about 18 million trips in the year 2019 that have at least one recorded point in a bounding box surrounding Frankfurt am Main. The map-matching results confirm an adequate performance of Valhalla map-matching, show a reduction of errors by distance calculation, and allow for further street-segment-based analysis.

Keywords GPS data · Trajectory · GPS error · Map-matching · Road network

Introduction

In recent years, location data have been collected more than ever. Most moving objects, especially cars, are now equipped with GPS loggers in one way or another. While new cars have mostly GPS trackers, old cars' movements are recorded using GPS-enabled smartphones. These data have been used increasingly in mobility research in past years. Although it has provided many new opportunities for research, it has also introduced challenges [30].

Tracking data are collected by sampling the GPS location of a moving object at a specific sampling rate (GPS points per unit of time). These tracking data, combined with other data collected by the object's sensors, such as speed, are called Floating Car Data (FCD). FCD is a powerful source in smart-mobility management systems to analyze and predict traffic speed on road networks and measure traffic

congestion. In addition, it is broadly used in research, for example, to find traffic patterns [21], to spot, explain, and predict accidents [20], and to infer travel mode [48].

GPS data are not accurate [5]. According to Plaudis et al. [35], a sequence of GPS points, called a GPS trajectory, is associated with two types of errors. First, measurement errors, i.e., the recorded location can deviate from the true location. This is due to noise from several sources when recording GPS data [19]. Second, sampling errors, which refer to lost information between the recorded points. Ignoring these errors in the data can lead to false analyses and misleading conclusions. Map-matching is an approach to minimize the errors in GPS data. It refers to matching the recorded points to a representation (usually in the form of a graph) of a road network. Figure 1 shows an example of map-matching.

Map-matching can either be offline or in real time [29]. Offline map-matching refers to matching the recorded coordinates after the trip is completed. Offline algorithms use all points of an entire trajectory to estimate the true positions on the road. These are primarily used when working with historical data. Real-time map-matching refers to matching the points during the trip as the points are being recorded. Obviously, real-time algorithms can

✉ Siavash Saki
Siavash.saki@fb3.fra-uas.de

Tobias Hagen
thagen@fb3.fra-uas.de

¹ Research Lab for Urban Transport, Frankfurt University of Applied Sciences, Frankfurt am Main, Germany

Fig. 1 GPS recorded points (red) and matched points (blue). The blue line shows the corresponding trajectory found by map-matching. basemap: ©OpenStreetMap ©CARTO



use only previously recorded coordinates to estimate the true position and direction on the road at a certain point. These are used mainly by navigation services and in autonomous vehicles.

A comparison between different map-matching algorithms and generally evaluating the map-matching quality is not straightforward [38], since the “true” positions of the recorded points are unknown. However, almost all the matching algorithms implemented by available map-matching services demonstrate good performance when the sampling rate is high, i.e., time interval between measured GPS points of less than 30 s [4–6].

This paper demonstrates a practical guide to open-source map-matching. The section “[Brief Overview of Available Approaches](#)” reviews the historical evolution of map-matching algorithms and classifies their applications. The currently most prominent map-matching algorithm, which is based on hidden Markov models, is briefly explained in the section “[Map-Matching Based on the HMM Algorithm](#)”. Next, several available map-matching service providers and open-source routing engines are compared in the section “[Available Map-Matching Services](#)”. We chose the open-source routing engine Valhalla between available options to demonstrate the map-matching. The section “[GPS Data Used](#)” describes the GPS dataset used. The section “[Building a Valhalla Routing Server](#)” explains shortly how a Valhalla engine can be built. The map-matching process, API parameters and data flows are demonstrated in the section “[Map-Matching Process](#)”. The section “[Results and Evaluation](#)” discusses the results and evaluate the map-matching performed by Valhalla. Finally, the section “[Conclusion](#)” concludes the paper.

Brief Overview of Available Approaches

Some early algorithms [25, 39, 41] tried statistical estimation for map-matching. Their approach was to fit a curve to recorded points and find the road segment that matches the curve better. This approach worked only for some special cases, but it did not function well in most practical applications. The most naïve algorithm for map-matching, which matches the points to the nearest street segments, is discussed by Bernstein et al. [3], Kim [23], White et al. [45], and Taylor et al. [42]. This algorithm is sensitive to outliers and does not use the previously recorded points. However, it is fast and can provide a baseline for more advanced map-matching algorithms. Bernstein et al. [3] and White et al. [45] added a simple improvement to this algorithm by taking the heading information of the GPS data into account. If the GPS heading does not correspond to the road heading, then the road is discarded. Another improvement to this algorithm is suggested and discussed by Yang et al. [47] and Quddus et al. [37]. They use a Kalman filter to drop unreasonable GPS data points which leads to better map-matching results. A revolutionary map-matching algorithm is proposed by Newson and Krumm [34] which is the basis for many further approaches. This algorithm is also implemented in many open-source routing engines (such as Valhalla) as well as commercial map-matching services. This method, which is based on hidden Markov models (HMM), finds the most likely sequence of road segments that match the sequence of GPS points.

The HMM has been widely used for map-matching applications and optimized for different settings and environments (see, e.g., [12, 24, 28, 40, 46]). An essential

aspect of map-matching using HMM is path-finding for estimating transition probabilities. HMM determines the shortest (least-cost) path between the two successive hidden states. This is a classic point2point shortest-path optimization problem (P2P), which has been extensively studied in the last decades (see, e.g., [1, 2, 8–10, 13, 22, 49]). There are exact and metaheuristics algorithms for solving the shortest-path problem. An example of an exact algorithm for finding the shortest path from a starting node S to a target node T in a weighted graph is the Dijkstra's algorithm [10]. This algorithm finds the shortest path from S to T by exploring the shortest path from S to all other vertices in the graph.

A heuristic extension of Dijkstra's algorithm is the A* search algorithm [16], which is implemented in Valhalla. Unlike Dijkstra, A* does not explore all routes from the source to the destination, and it only explores the promising routes. It minimizes the sum of the cost from the source to the current node and the current node to the destination. While the cost of the path from the source to the current node is known, the cost of the path from the current node to the destination is estimated using a heuristic function such as Euclidian or Manhattan distance. In recent years, heuristic approaches have gained significant attention for solving shortest-path problems [31]. For example, in Chen et al. [7], a heuristic method is proposed, which is an extension of Dijkstra's algorithm, for finding the shortest path in traffic networks. The shortest-path problem is also relevant in other contexts rather than mobility. Hasan et al. [17] propose a heuristic genetic algorithm for finding the shortest path for Internet routing.

There are several literature reviews on map-matching attempting to classify existing methods. Quddus et al. [36] group the map-matching algorithms into geometric, topological, probabilistic, and advanced. Geometric map-matching is only based on geometric distance (e.g., point-to-curve), topological map-matching uses the contiguity of the roads, probabilistic methods approximate an error region using the uncertainty of position, and advanced map-matching is a combination of methods. In addition, this paper evaluates several available methods, and identify their constraints and limitations. Finally, it argues that the main problems are associated with the initial position identification. Wei et al. [44] review the map-matching algorithms and classify them as incremental max-weight, global max-weight, and global geometric. Max-weight methods integrate several factors (e.g., directions of recorded points and road, distance of the recorded point to road segments, and shortest path between recorded points) and select the candidate sequence with the highest score. Incremental max-weight only use previously recorded points for real-time applications and global max-weight use the entire trajectory for offline applications. For example, all algorithms using

hidden Markov models (e.g., Newson and Krumm [34] fall under global max-weight. Global geometric methods refer to models which match the points only based on geometric measures. Hashemi and Karimi [18] provide an overview of map-matching methods mainly for navigational (real-time) applications and are therefore not discussed further. Kubicka et al. [27] divide map-matching algorithms based on their application. The intention for this division is that map-matching problems vary across applications, and different methods could be suitable for different applications. While real-time algorithms are suitable for navigation purposes, offline algorithms are used in surveying applications. A most recent review of map-matching algorithms is conducted by Chao et al. [6]. They argue that previous categorizations of map-matching algorithms are outdated and are not useful anymore. Geometric algorithms are not implemented and recent hidden Markov models map-matching algorithms can be used in both real-time and offline applications. Therefore, they provide a new categorization of map-matching models according to their algorithms and applications. Moreover, by evaluating several map-matching methods, they find that both too high and too low sampling rates can be problematic for map-matching models.

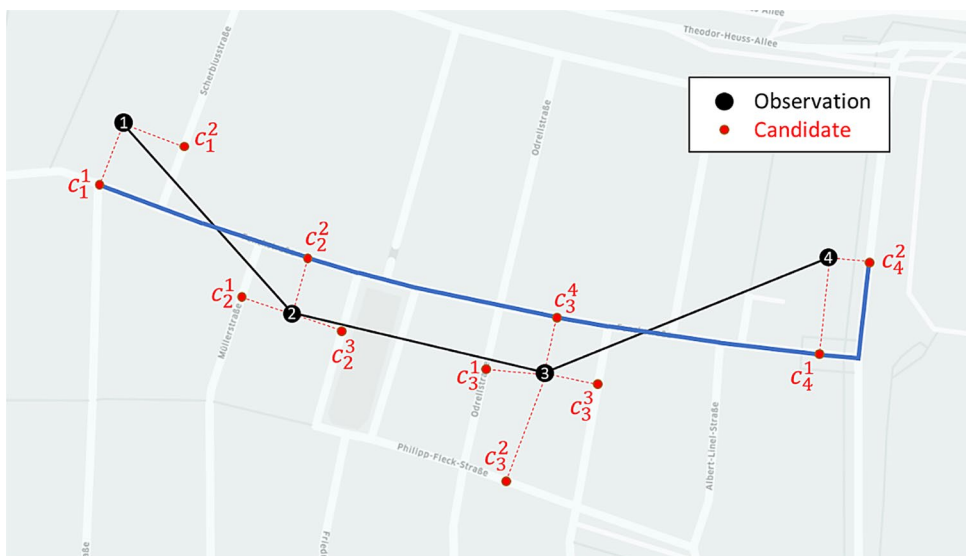
Map-Matching Based on the HMM Algorithm

Many map-matching service providers, such as Valhalla, Mapbox, and GraphHopper, use the HMM algorithm based on Newson and Krumm [34]. This algorithm, which is proposed by Microsoft, is also widely used by other companies such as Uber. In this approach, the problem is formulated as follows: given a sequence of (time-stamped) GPS points, each point has to match a road segment, what is the most likely sequence of road segments that match the sequence of GPS points? This formulation naturally fits the HMM, as the recorded GPS points (observations) refer to state measurements and the individual road segments refer to hidden states of the HMM algorithm. The most likely path is the sequence of states with the most likely transitions, where the transitions between road segments are ruled by connectivity in the road network.

The HMM algorithm works as follows: First, for each GPS point, the model finds the possible road segment matches, the so-called candidates. Figure 2 shows an example of a sequence of four GPS observations and their identified road segment candidates. These candidates are found within a certain search radius from the observation. Each observation has a set of candidates; for example, observation P_1 has two candidates, c_1^1 and c_1^2 .

Second, the measurement probability and the transition probabilities are calculated. The measurement probability is the likelihood of observing a particular measurement using

Fig. 2 GPS points (observations) and their corresponding road candidates. ©OpenStreet-Map ©CARTO



only that measurement, assuming that GPS noise is Gaussian distributed with a zero mean. This way, the candidates being further away from the observation are less likely to be considered. The transition possibility of two successive candidates gives the likelihood that the vehicle drove between them. For example, some transitions which require complex maneuvers are less likely to be considered. Moreover, the algorithm favors the transitions, with a great-circle distance¹ being about the same as the driving distance. Finally, considering both measurement and transition probabilities, the optimal path is identified. Figure 3 shows all the possible transitions between the road candidates for the GPS sequence of Fig. 2. The selected candidates and the optimal path are marked red.

There are different algorithms to find the optimal path. A naive approach is a brute-force solution that explores all the possible transitions. This could be associated with a long computation time. A relatively faster approach is the Viterbi algorithm [11], which is used in the Newson and Krumm [34] approach. This algorithm also calculates the shortest path for all the consecutive pairs of nodes. Assuming a sequence has M measurements with an average of T states per measurement, the algorithm must calculate $M \cdot T \cdot T$ shortest paths, corresponding to the time complexity of $O(N^2T)$. This calculation can also take much time, especially in dense urban areas with many possible transitions. Another method to reduce the number of shortest-path calculations is to use Dijkstra’s algorithm, which is implemented in Valhalla. It uses a greedy algorithm to extract only the most likely nodes at each state that could lead to a lower runtime than Viterbi.

Available Map-Matching Services

There are many map service providers which offer map-matching. Several examples and their API pricing are listed in Table 1. These services provide ease-of-use and fast implementation for applications, which require map-matching. For a few trips, map-matching is cheap, but costs for matching a large number of trajectories can become exorbitant. There are also limitations in each service. For example, in a single request, a maximum number of 100 locations is allowed while using Google Maps API or Mapbox API.

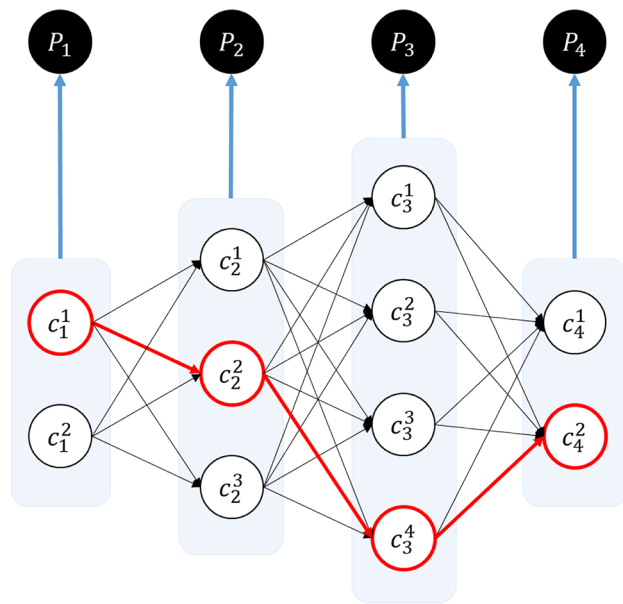


Fig. 3 Possible transitions between states and optimal path

¹ The shortest distance between two points on a sphere.

Table 1 Map-matching API pricing of several map service providers

	Price/request ^a (€)	Price/1,000,000 requests ^b (€)	Request locations limitation	Other requests ^c limitations
Google Maps [14]	0.010	10,000	100 points/require	
GraphHopper [15]	0.003	3,000	500 points/require	1500 requests/day
Mapbox [32]	0.0016	1,600	100 points/request	300 requests/minute
Mapzen [33]	0.001	1,000		

^aA request with maximum number of locations

^bPrices do not linearly increase. There are usually custom plans for heavy clients with discounts. This is just for a quick comparison between the services

^cRequests with maximum number of locations

Table 2 Comparison between OSRM and Valhalla routing engines according to Kreiser [26]

	OSRM	Valhalla
Pros	Fast performance Support multi-mode travel	Low memory usage Provides different routes at runtime by varying request parameters
Cons	High memory usage Precomputing routes do not provide different routes at runtime (e. g., avoid tolls, penalize ferries, etc.)	Slower performance than OSRM Localization is not as advanced as OSRM

This could be a problem for long trajectories which contain many location points or are recorded at a high sampling rate, since splitting trajectories increases the complexity of data engineering and reduces the quality of map matching results. Further limitations in API usage, such as daily or minutely requests restrictions, are also present. For example, GraphHopper and Mapbox allow only 1500 requests per day and 300 requests per minute, respectively. These limitations could be acceptable for implementing these services in applications like smartphone apps. There are usually custom pricing plans for clients with scaling businesses. However, these limitations could be obstructive when working with historical data that require map-matching for a large number of trajectories at once.

Another way to conduct map-matching is to build a routing server capable of map-matching. Fortunately, there are several open-source routing engines that provide routing services, including map-matching. Two widely used open-source engines are OSRM² and Valhalla.³ Some of the advantages and disadvantages of these two routing engines, as discussed by Kreiser [26], are listed in Table 2.

These engines have a high performance and are vastly used by large enterprises. Some of the above-mentioned service providers use these engines under the hood. OSRM and Valhalla are used (and also mainly developed and maintained) by Mapbox and Mapzen, respectively. Another

advantage of both these engines is that they use OpenStreet-Map data, making it easy for further analysis and visualization purposes. Here, we chose to build a Valhalla routing engine, since it needs significantly lower memory. A comprehensive study on map-matching using OSRM is performed by Vander Laan et al. [43]. They propose a scalable well-constructed enhancement framework for GPS data that could map-match millions of trajectories.

Other examples for map-matching frameworks are pgMapMatch⁴ and Fast Map-Matching (FFM).⁵ However, as the advantages of OSRM and Valhalla outweigh the other frameworks, they are not discussed here.

GPS Data Used

The GPS data consist of about 18 million trips with more than 5 billion GPS observations (locations with timestamp). These trips have at least one GPS location within a bounding box surrounding Frankfurt am Main in the year 2019. The data source is INRIX, a private company that provides location-wise data and analytics. The journeys are recorded using either a smartphone or embedded GPS devices. The dataset contains private as well as fleet trips, and it is distinguished between vehicle weight classes. Light, medium, and heavy vehicles range from 0 to 14,000 lb, 14,000 lb to

² <http://project-osrm.org/>.

³ <https://github.com/valhalla/valhalla>.

⁴ <https://github.com/amillb/pgMapMatch>.

⁵ <https://github.com/cyang-kth/fmm>.

Table 3 Number of trips in each category regarding source, vehicle weight, and type

Source	Weight class	Type	Trips count
Embedded GPS	Light	Consumer	681,699
		Fleet	364
	Medium	Fleet	2,776,388
Mobile device	Heavy	Fleet	153,223
	Light	Consumer	14,160,423
		Fleet	17,508

Fig. 4 Waypoints of 100,000 trips visualized on the map. ©OpenStreetMap ©CARTO

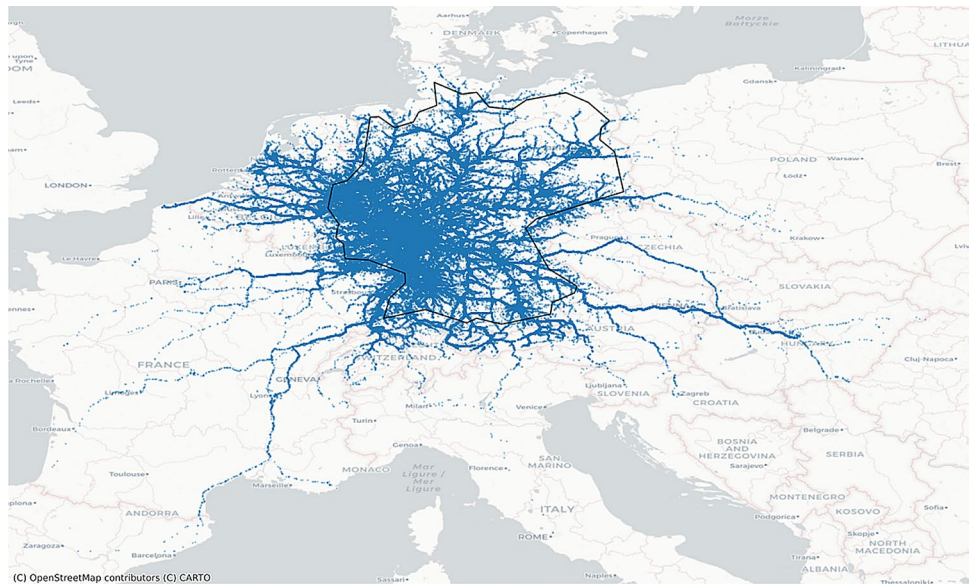


Table 4 The descriptive statistics of trips' mean and max speed, length, and sampling frequency of the whole dataset

	Mean speed (Kph)	Max speed (Kph)	Trip length (m)	Sampling frequency (S)
Mean	69.03	142.84	96,773.07	45.7
Std	34.53	63.81	124,566.1	57.6
Min	0.03	0.07	0.42	0.5
25%	39.22	92.03	10,572.04	6.9
50%	71.35	146.49	39,752.61	17.7
75%	97.2	197.04	144,949.5	57.9
max	199.99	250	2,216,875	985.2

26,000 lb, and higher than 26,000 lb, respectively. An overview of the number of trips in each category can be seen in Table 3.

The data are stored in compressed GZIP format on the AWS S3 storage and takes more than 300 GB. The waypoints are separated monthly. Each month has about 80 GZIP part files containing the waypoints. A 0.56% sample consisting of 100,000 trips is visualized in Fig. 4.

A description of the trips' metadata can be seen in the Table 4.

Building a Valhalla Routing Server

There are two ways to install Valhalla. The first one is to build it from the source. The second one is to run a Valhalla instance using Docker. The second option is less complex and is more efficient with regard to time and resources. The requirement is the latest docker and docker-compose running

on an Ubuntu 20.04. It is recommended to run the Valhalla Docker image by GIS OPS⁶, since it is more straightforward than the original Valhalla Docker image.

To build the Valhalla Docker with Germany's tiles, the docker-compose.yml file should be properly configured. The

⁶ <https://github.com/gis-ops/docker-valhalla>.

Fig. 5 Preprocessing step: only the waypoints in Germany are kept

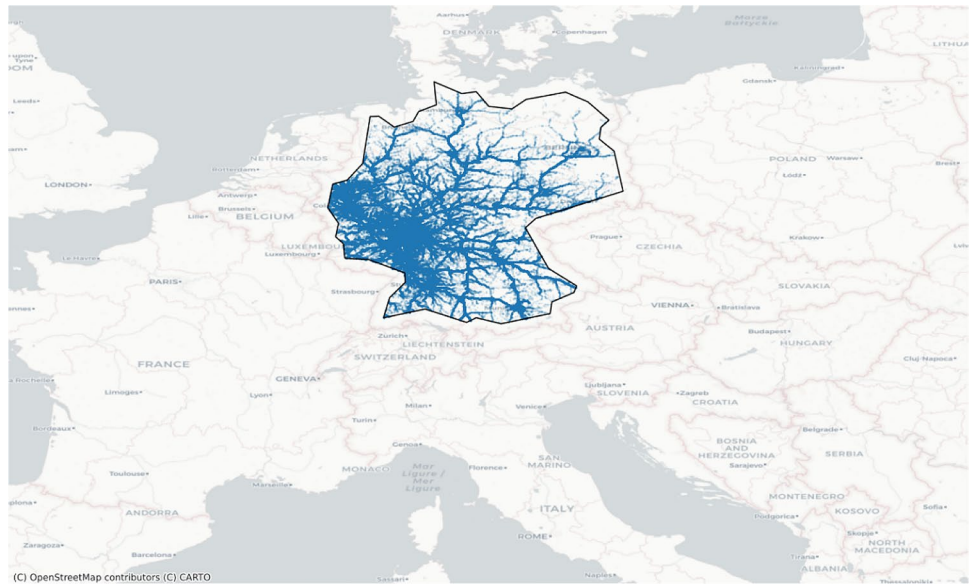
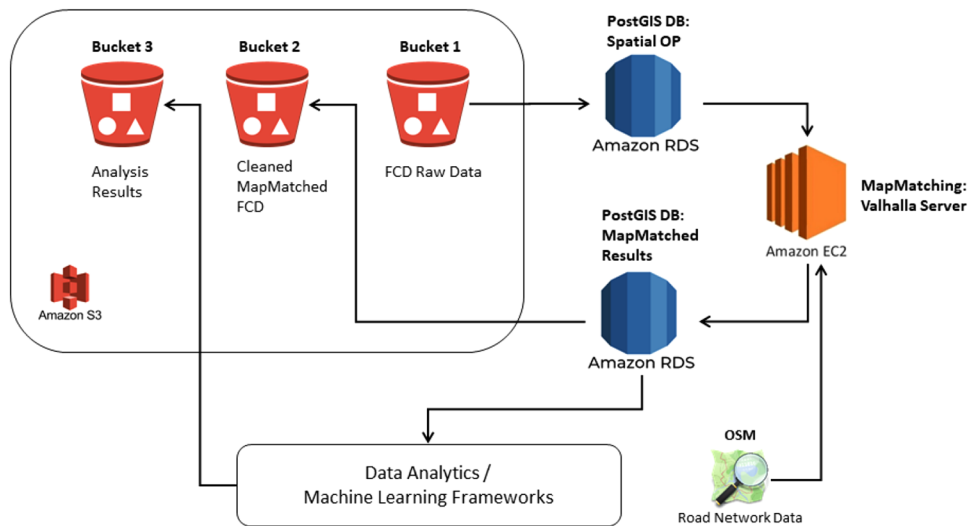


Fig. 6 Map-matching data pipeline



tiles data are available on geofabrik,⁷ which is an official member of OpenStreetMap. After integrating the geofabrik link⁸ in the YAML file, Germany’s tiles could be extracted directly from a PBF file while building Valhalla. An example of a YAML configuration file is shown in appendix.

Depending on the hardware, it can take several hours to build Valhalla completely with the Germany’s tiles. In our case, it took more than 4 h on a t2.2xlarge AWS EC2 instance with 32 GB RAM and 8 CPU cores. We store this server as an Amazon Machine Image (AMI) to make it reusable. Therefore, each time we need a map-matching server, we can simply start an EC2 instance with this AMI. We can

even run multiple map-matching servers in parallel by starting several servers using this AMI.

Map-Matching Process

The first challenge with our dataset is assembling the journeys’ trajectories, since the waypoints are stored in many different files without any logical order. There are several different tools for big data analytics. Since our data are spatial and we need to perform spatial operations on the data, we chose the PostGIS⁹ spatial database. PostGIS is an extension

⁷ <https://www.geofabrik.de/>.

⁸ <http://download.geofabrik.de/europe/germany-latest.osm.pbf>.

⁹ <https://postgis.net/>.

Table 5 Examples of map-matching hyperparameters

Parameter	Description	Default
Sigma_z	To specify the GPS accuracy	4.07
Beta	To weigh the transition cost	3
Search_radius	search radius for road candidates	50 m

to PostgreSQL, which adds support for geographic objects and allows to run spatial operations using SQL.

The first step is to import all the GZIP waypoint files from the S3 storage into a PostGIS server. Any required preprocessing step can be done on this server. For example, we drop all the GPS points outside Germany, since they are not of our interest for map-matching purposes. Figure 5 shows the same sample visualized in Fig. 4 after this preprocessing step.

The next step is assembling the trajectories. We group the points by their TripID, order the points by their timestamp and store them in a LineString geometry type. Afterward, they are ready to be sent to the map-matching server. Figure 6 shows the data flow in our map-matching architecture. For each trip, a request is generated and sent to the map-matching server or, more precisely, to Meili using the Library API. Meili is a namespace within Valhalla that includes the map-matching code and is responsible for map-matching functionality.

We can control the performance and accuracy of Meili by adjusting the map-matching hyperparameters. Some of the parameters are listed in Table 5.¹⁰ sigma_z is the standard deviation of the GPS measurement error and represents the GPS noise. It directly affects the emission probabilities. beta weighs the transition costs and affects the transition probability of two successive points. search_radius could affect the number of initial candidates that the HMM considers. The default values for these parameters in Valhalla are set regarding the recommendations from Newson and Krumm [34]. For instance, if the GPS data exhibit a lot of noise, we can increase the search_radius, beta, and sigma_z. This could increase the runtime as it considers more measurements and transition possibilities, but it could lead to better results depending on the data. There are also further parameters to set up the map-matching environment. For example, “transport mode” can be chosen between “auto”, “bicycle”, “pedestrian”, and “multimodal”.

Integrating the timestamps of the coordinates in the API request is optional. Timestamps affect the calculated transition probabilities. The algorithm favors the transitions with durations (calculated by the shortest-path algorithm) being

Table 6 The descriptive statistics of trips’ mean and max speed, length, and sampling frequency of the sample used

	Points count	Trip length (KM)	Sampling frequency (S)
Mean	302.22	98.999	46.8
Std	545.47	122.181	55.8
Min	2	0	0.5
25%	25	11.27	8.3
50%	95	41.79	22.1
75%	311	151.91	61
Max	10,578	1365.38	985.2

Table 7 Range of hyperparameters chosen for the experiment

Parameter	Min	Max	Steps	Scale
Sigma_z	0.5	12	8	Linear
Beta	0.5	12	8	Linear
Search_radius	20	100	9	Linear

similar to the actual duration. This can enhance the map-matching performance.

Meili processes the request, performs the map-matching based on OSM tiles, and creates a response in JSON format. While parsing the response, we can split the result into three parts: matched points, trajectory, and narrative. Matched points are the corresponding locations of the recorded points on the road network. Trajectory is a sequence of corresponding road segments in a polyline format (convertible to LineString). It can be used to calculate the trip distance or to visualize the trip on the map. Narrative contains the maneuver points and their details (e.g., type of turn or verbal instruction by a navigation app) based on the actual path.

The parsed and processed responses are then directed to another PostGIS database responsible for holding the results. When the map-matching is finished for all the GPS points, we can export the results from the PostGIS database to the S3 storage or to send them to the machine learning data pipeline to do further analysis.

Results and Evaluation

We tested our map-matching environment with a sample of 1,216,476 trajectories. A description of the sample dataset can be seen in Table 6.

In our Valhalla’s map-matching environment, the “transport mode” is set to “auto”. All other parameters are set to default. The first attempt succeeds in map-matching 78.24% of the journeys ($n = 951,804$). After exploring the

¹⁰ A complete list of parameters can be found at: <https://valhalla.readthedocs.io/en/latest/meili/configuration/>.

Fig. 7 Trajectories before map-matching

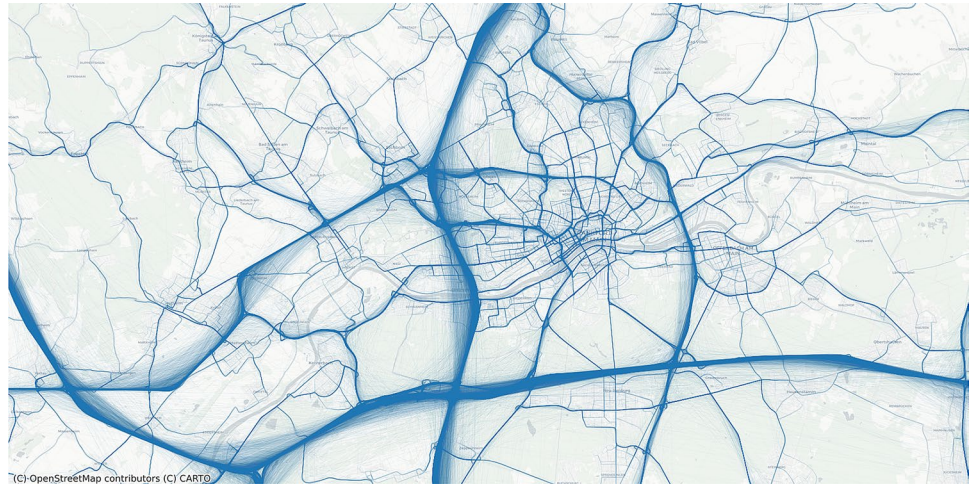


Fig. 8 Trajectories after map-matching



unsuccessful cases, we could establish two systematic failures associated with two clusters of journeys: (1) journeys longer than 200 km and (2) journeys with too few GPS measurements. The reason why the journeys longer than 200 km could not be map-matched is unknown; however, a possible limitation might be implemented in Valhalla without being documented. After discovering this issue, the journeys with a trip length of more than 200 km are split into several parts, so that each part is less than 200 km. The map-matching process is conducted again, and the new success rate is 95.20% ($n = 1,158,144$).

To discover if any better success rate could be achieved, a grid search is done on a random sample of 5000 journeys. The selected hyperparameters, resulting in 556 ($= 8 * 8 * 9$) combinations, for the grid search are listed in Table 7.

The grid search results demonstrate that the success rate of map-matching is constant at 95.26% for all the hyperparameter combinations. This means a better result on this sample could not be achieved with the selected finite subset

from the hyperparameters' domain space. This could be due to the low number of GPS measurements and high noise in the “unsuccessful” journeys. However, we cannot make any claim about the success rate of the map-matching of our sample for the combinations of hyperparameters that fall out of the selected grid search.

The raw data before map-matching is visualized in Fig. 7, and the trajectories after map-matching are visualized in Fig. 8. It can be seen that before map-matching, the trajectories can deviate from roads, i.e., they are associated with errors. After map-matching, they perfectly lie on the road network.

We calculate the trip's distance before and after map-matching to get an overview of the changes. A sample of 20,000 trips is chosen containing trips with a distance of more than 1000 m and a continuous LineString from the origin to the destination after map-matching. The trip distance is defined as the sum of the great-circle distances between the location points. The absolute and relative difference in

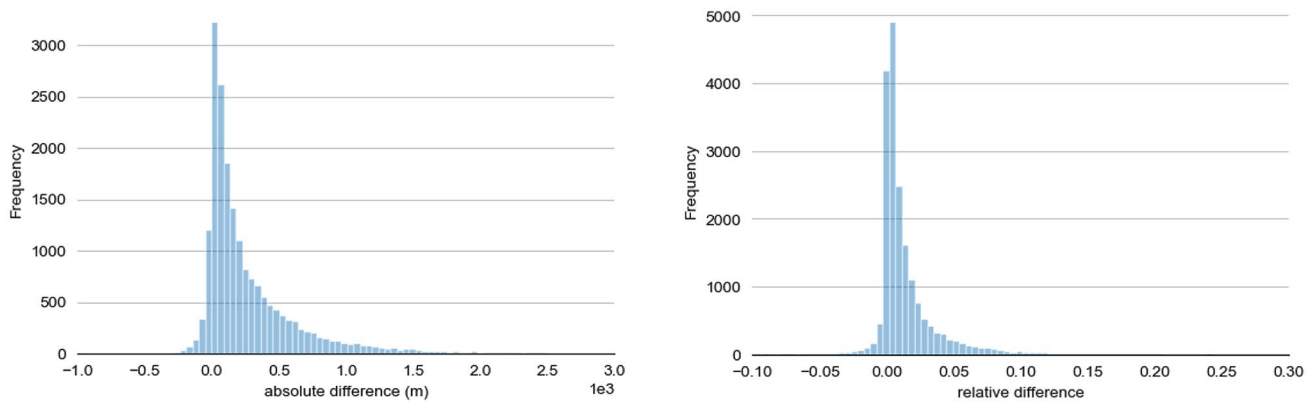


Fig. 9 Absolute (left) and relative difference (right) of trip distance after map-matching

Table 8 Average runtime of map-matching for each trip separated by different length-points count groups

	Trip length [TL] (in kilometers)		
	TL < 50	50 ≤ TL ≤ 200	TL > 200 ^a
Points count [PC]			
PC < 50	0.0353 s (<i>n</i> = 363,005)	0.0446 s (<i>n</i> = 41,088)	0.0599 s (<i>n</i> = 1237)
50 ≤ PC ≤ 100	0.0468 s (<i>n</i> = 108,004)	0.0576 s (<i>n</i> = 41,644)	0.0769 s (<i>n</i> = 15,331)
PC > 100	0.0562 s (<i>n</i> = 226,881)	0.0828 s (<i>n</i> = 171,199)	0.1493 s (<i>n</i> = 189,755)

^aFor every trip with a TL > 200, the sum of the runtimes of its parts is considered

trip distance after map-matching in comparison to the original trip distance can be seen in Fig. 9. The trip distance after map-matching is on average 1.4% longer than the original trip distance calculated from raw data. One reason for this increase is the fact that curves are mapped more realistic after map-matching (see Fig. 1).

To provide an overview of map-matching runtime, the duration of map-matching of every journey is measured. Table 8 shows the average map-matching runtime for different groups of journeys based on the number of GPS measurements and trip length. We can see that the average runtime increases by increasing the points count and trip length. This is intuitive as the number of computational tasks increases.

Map-matching runtime depends on the hardware. For example, in this case, the map-matching server is set up on an r5a.2xlarge EC2 instance. This instance has 64 GB RAM and 8 vCPUs and costs \$0.548 per hour on-demand on a Linux machine at the time this paper is written. This indicates a relatively low cost for this map-matching approach.

Conclusion

In recent years, there has been a drastic increase in using GPS data in mobility research and smart-mobility applications. Moreover, much research has been done on dealing

with inaccuracy in the GPS movement data. Against this background, map-matching has been recognized to be an essential preprocessing step for minimizing errors.

Using map-matching, we can find the road segments matched to the recorded points of a GPS trajectory, allowing further analysis based on road segments. There are several map service providers that offer map-matching. However, their APIs are usually restricted (e.g., number of locations per request) and costly, which make them not suitable for map-matching historical big GPS data. An alternative to these services is building a map-matching server using open-source routing engines such as Valhalla.

We built a cloud-based map-matching framework on AWS. Within this framework, PostGIS is used to perform spatial operations and to assemble trajectories, and Valhalla runs on the map-matching server. We tested Valhalla with a sample of about 1.2 million GPS trajectories and 95.20% of the journeys were successfully map-matched. Other journeys (the remaining 4.80%) contained too few GPS measurements to yield a reliable map-matching result. This proves that Valhalla can be considered a powerful tool for map-matching big GPS datasets regarding cost, runtime, and performance. A visualization of map-matched trips shows that their trajectories perfectly lie on the road network. The trajectory distance calculated after map-matching is on average 1.4% longer than the distance calculated using raw data before map-matching. Assuming that map-matching reveals the

actual driven route, it leads to a significant error reduction while calculating trajectory distance. This could improve the accuracy of further analyses that use trip distance.

Map-matching using the suggested framework by Valhalla is fast enough for most applications, since it is supposed to be used for offline and not for real-time map-matching. However, it can still take significant time. Alternatives, such as OSRM, are relatively faster and could replace Valhalla when the overall runtime is an essential aspect. With the parallelization of Valhalla servers, the overall map-matching runtime could be reduced linearly (which could be associated with more infrastructure and cost). Overall, the proposed map-matching framework is efficient and scalable and can be used to conduct map-matching for terabytes of geospatial data.

Some of the limitations of our work are as follows: (1) We cannot make any claim about the accuracy of the results, since the true positions of raw GPS measurements are unknown. A possible approach to evaluate the accuracy of the results would be testing the framework with artificial GPS points and a deviation metric. (2) Because of the computation limits, only a small subset of hyperparameters are chosen for the grid search. The performance of the algorithm outside of this space is not studied.

Future work should use other open-source alternatives, such as OSRM. It should demonstrate how OSRM could be built and used for map-matching. Using the same dataset, its performance should be evaluated and a comparison between the open-source routing engines, i.e., OSRM and Valhalla, could help to decide which service to use depending on the application. In addition, using an artificial GPS dataset, the accuracy of both frameworks should be measured and compared.

Appendix

To build the Valhalla Docker with Germany's tiles, the docker-compose.yml file can be configured as follows:

```

services:
  valhalla:
    image: gisops/valhalla:latest
    ports:
      - "8002:8002"
    volumes:
      - ./custom_files:/custom_files
    environment:
      - tile_urls=http://download.geofabrik.de/europe/germany-latest.osm.pbf
      - min_x=5.8 # Germany's minimum longitude
      - max_x=15.1 # Germany's maximum longitude
      - min_y=47.2 # Germany's minimum latitude

```

```

- max_y=55.2 # Germany's maximum latitude
- use_tiles_ignore_pbf=True
- build_time_zones=True
- build_elevation=True
- build_admins=True
- force_rebuild_elevation=False
- force_rebuild=False

```

Using this configuration, the local PBF files, located in custom_files folder, are prioritized. To get the Germany's tiles using the specified link, this folder must be empty. After configuring the YAML file, Valhalla can be built by: `docker-compose up --build`

Once the installation is finished, Valhalla should be running on <https://en.wikipedia.org/wiki/Localhost> .

Funding Open Access funding enabled and organized by Projekt DEAL. This work is funded by the state of Hesse and HOLM funding under the "Innovations in Logistics and Mobility" measure of the Hessian Ministry of Economics, Energy, Transport and Housing [HA Project No.: 1017/21-19].

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Aljazzar H, Leue S. K*: a heuristic search algorithm for finding the k shortest paths. *Artif Intell.* 2011;175(18):2129–54. <https://doi.org/10.1016/j.artint.2011.07.003>.
2. Beke L, Weiszner M, Chen J. A comparison of genetic representations and initialisation methods for the multi-objective shortest path problem on multigraphs. *SN Comput Sci.* 2021. <https://doi.org/10.1007/s42979-021-00512-z>.
3. Bernstein D, Kornhauser A, New Jersey Institute of Technology. An Introduction to Map Matching for Personal Navigation Assistants: New Jersey TIDE Center. 1966. <https://rosap.nrl.navy.mil/view/doc/38257>.
4. Chao P. A Study on map-matching and map inference problems. Queensland: University of Queensland Library; 2020.

5. Chao P, Hua W, Mao R, Xu J, Zhou X. A survey and quantitative study on map inference algorithms from GPS Trajectories. *IEEE Trans Knowl Data Eng.* 2020. <https://doi.org/10.1109/TKDE.2020.2977034>.
6. Chao P, Ye Xu, Hua W, Zhou X. A survey on map-matching algorithms. In: Renata B-G, Jianzhong Q, Weiqing W, editors. *Databases theory and applications*. Cham: Springer International Publishing; 2020. p. 121–33 (Lecture Notes in Computer Science).
7. Chen X, Fei Q, Li W. A new shortest path algorithm based on heuristic strategy. *World Congr Intell Control Autom.* 2006;1:2531–6. <https://doi.org/10.1109/wcica.2006.1712818>.
8. Cherkassky BV, Goldberg AV, Radzik T. Shortest paths algorithms: theory and experimental evaluation. *Math Program.* 1996;73(2):129–74. <https://doi.org/10.1007/BF02592101>.
9. Di Caprio D, Ebrahimnejad A, Alrezaamiri H, Santos-Arteaga FJ. A novel ant colony algorithm for solving shortest path problems with fuzzy arc weights. *Alex Eng J.* 2022;61(5):3403–15. <https://doi.org/10.1016/j.aej.2021.08.058>.
10. Dijkstra, EW. A note on two problems in connexion with graphs. In Edsger Wybe Dijkstra: His Life, Work, and Legacy, 2022. pp. 287–290.
11. Forney G, David JR (2005): The viterbi algorithm: a personal history. 2022. <https://arxiv.org/pdf/cs/0504020>.
12. Fu X, Zhang J, Zhang Y. An online map matching algorithm based on second-order hidden Markov model. *J Adv Transp.* 2021. <https://doi.org/10.1155/2021/9993860>.
13. Goldberg AV, Silverstein C. Implementations of Dijkstra's algorithm based on multi-level buckets. In: Donald WH, William WH, Panos MP, editors. *Network optimization*. 450th ed. Berlin: Springer; 1997. p. 292–327 (Lecture Notes in Economics and Mathematical Systems).
14. Google Maps Platform. Pricing plans and API costs-google maps platform. 2022. <https://mapsplatform.google.com/pricing/>. Accessed 02 July 2022.
15. GraphHopper. Pricing-GraphHopper Directions API. 2022. <https://www.graphhopper.com/pricing/>. Accessed 02 July 2022.
16. Hart P, Nilsson N, Raphael B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans Syst Sci Cyber.* 1968;4(2):100–7. <https://doi.org/10.1109/TSSC.1968.300136>.
17. Hasan BS, Khamees MA, Mahmoud ASH. A heuristic genetic algorithm for the single source shortest path problem. *IEEE/ACS Int Conf.* 2007. <https://doi.org/10.1109/aiccsa.2007.370882>.
18. Hashemi M, Karimi HA. A critical review of real-time map-matching algorithms: current issues and future directions. *Comput Environ Urban Syst.* 2014;48:153–65. <https://doi.org/10.1016/j.compenvurbusys.2014.07.009>.
19. Hendawi A, Shen J, Sabbineni SS, Song Y, Cao P, Zhang Z, et al. Noise patterns in GPS trajectories. *IEEE Int Conf Mob Data Manag (MDM).* 2020. <https://doi.org/10.1109/mdm48529.2020.00040>.
20. Hunter M, Saldivar-Carranza E, Desai J, Mathew JK, Li H, Bullcock DM. A proactive approach to evaluating intersection safety using hard-braking data. *J Big Data Anal Transp.* 2021;3(2):81–94. <https://doi.org/10.1007/s42421-021-00039-y>.
21. Karve V, Yager D, Abolhelm M, Work DB, Sowers RB. Seasonal disorder in urban traffic patterns: a low rank analysis. *J Big Data Anal Transp.* 2021;3(1):43–60. <https://doi.org/10.1007/s42421-021-00033-4>.
22. Khani A, Boyles SD. An exact algorithm for the mean–standard deviation shortest path problem. *Transp Res Part B.* 2015;81:252–66. <https://doi.org/10.1016/j.trb.2015.04.002>.
23. Kim J-S. Node based map matching algorithm for car navigation system. In: *International Symposium on Automotive Technology and Automation (29th : 1996 : Florence, Italy)*. Global deployment of advanced transportation telematics/ITS, 1996.
24. Koller H, Widhalm P, Dragaschnig M, Graser A. Fast hidden markov model map-matching for sparse and noisy trajectories. In: *2015 IEEE 18th International Conference on Intelligent Transportation Systems*. 2015 IEEE 18th International Conference on Intelligent Transportation Systems - (ITSC 2015). Gran Canaria, Spain, IEEE, 2015, pp. 2557–61.
25. Krakiwsky EJ, Harris CB, Wong RVC. A Kalman filter for integrating dead reckoning, map matching and GPS positioning. In: *IEEE PLANS '88., Position Location and Navigation Symposium, Record. 'Navigation into the 21st Century'*. IEEE PLANS '88., Position Location and Navigation Symposium, Record. 'Navigation into the 21st Century'. Orlando, FL, USA, IEEE, 1988, pp. 39–46.
26. Kreiser K. OSRM vs Valhalla. 2018. <https://github.com/valhalla/valhalla/issues/1514#issuecomment-419160356>. Accessed 02 July 2022.
27. Kubicka M, Cela A, Mounier H, Niculescu S-I. Comparative study and application-oriented classification of vehicular map-matching methods. *IEEE Intell Transport Syst Mag.* 2018;10(2):150–66. <https://doi.org/10.1109/MITS.2018.2806630>.
28. Luo A, Chen S, Xv B. Enhanced map-matching algorithm with a hidden markov model for mobile phone positioning. *IJGI.* 2017;6(11):327. <https://doi.org/10.3390/ijgi6110327>.
29. Luo L, Hou X, Cai W, Guo B. Incremental route inference from low-sampling GPS data: an opportunistic approach to online map matching. *Inf Sci.* 2020;512:1407–23. <https://doi.org/10.1016/j.ins.2019.10.060>.
30. Lyu C, Wu X, Liu Y, Liu Z. A partial-Fréchet-Distance-Based Framework For Bus Route Identification. *IEEE Trans Intell Transport Syst.* 2021. <https://doi.org/10.1109/TITS.2021.3069630>.
31. Magzhan K, Hajar MJ. A review and evaluations of shortest path algorithms. In: *International Journal of Scientific and Technology Research*, 2013, pp. 99–104.
32. Mapbox. Mapbox pricing. 2022. <https://www.mapbox.com/pricing/#matching>. Accessed 2 July 2022.
33. Mapzen. Mapzen map matching. 2022. <https://www.mapzen.com/products/mobility/map-matching/>. Accessed 2 July 2022.
34. Newson P, Krumm J. Hidden Markov map matching through noise and sparseness. In: Wolfson O, Agrawal D, Chang-Tien L, editors. *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems - GIS '09*. The 17th ACM SIGSPATIAL international conference. New York: ACM Press; 2009. p. 336.
35. Plaudis M, Azam M, Jacoby D, Drouin M-A, Coady Y. An Algorithmic Approach to Quantifying GPS Trajectory Error. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 3909–16.
36. Quddus MA, Ochieng WY, Noland RB. Current map-matching algorithms for transport applications: state-of-the-art and future research directions. *Transp Res Part C.* 2007;15(5):312–28. <https://doi.org/10.1016/j.trc.2007.05.002>.
37. Quddus MA, Ochieng WY, Zhao L, Noland RB. A general map matching algorithm for transport telematics applications. *GPS Solut.* 2003;7(3):157–67. <https://doi.org/10.1007/s10291-003-0069-z>.
38. Rehrl K, Gröchenig S, Wimmer M. Optimization and evaluation of a high-performance open-source map-matching implementation. In: Mansourian A, Pilesjö P, Harrie L, van Lammeren R, editors. *Geospatial technologies for all*. Cham: Springer International Publishing; 2018, p. 251–70 (Lecture Notes in Geoinformation and Cartography).
39. Scott CA, Drane CR. Increased accuracy of motor vehicle position estimation by utilising map data: vehicle dynamics, and other information sources. In: *Proceedings of VNIS'94 - 1994 Vehicle Navigation and Information Systems Conference*. VNIS'94 - 1994

- Vehicle Navigation and Information Systems Conference. Yokohama, Japan, , IEEE, 1994, pp. 585–90.
40. Szwed P, Pekala K. An incremental map-matching algorithm based on hidden Markov model. In: Rutkowski L, Korythowski M, Scherer R, Tadeusiewicz R, Zadeh LA, Zurada JM, editors. Artificial intelligence and soft computing. 13th International Conference, ICAISC 2014, Zakopane, Poland, June 1-5, 2014, proceedings. 8468th ed. Berlin: Springer; 2014. p. 579–90 (LNCS sublibrary. SL 7, Artificial intelligence, 8468).
 41. Takashi JO, Miki H, Hideo K. a map matching method with the innovation of the Kalman filtering. IEICE Trans Fundam Electr Commun Comput Sci. 1996;E79-A(11):1853–5.
 42. Taylor G, Blewitt G, Steup D, Corbett S, Car A. Road reduction filtering for GPS-GIS navigation. Trans GIS. 2001;5(3):193–207. <https://doi.org/10.1111/1467-9671.00077>.
 43. Vander Laan Z, Franz M, Marković N. scalable framework for enhancing raw GPS trajectory data: application to trip analytics for transportation planning. J Big Data Anal Transp. 2021;3(2):119–39. <https://doi.org/10.1007/s42421-021-00040-5>.
 44. Wei H, Wang Y, Forman G, Zhu Y. Map matching by Fréchet distance and global weight optimization. Technical Paper, Department of Computer Science and Engineering, 2013, p.19.
 45. White CE, Bernstein D, Kornhauser AL. Some map matching algorithms for personal navigation assistants. Transp Res Part C. 2000;8(1–6):91–108. [https://doi.org/10.1016/S0968-090X\(00\)00026-7](https://doi.org/10.1016/S0968-090X(00)00026-7).
 46. Yang C, Gidófalvi G. Fast map matching, an algorithm integrating hidden Markov model with precomputation. Int J Geogr Inf Sci. 2018;32(3):547–70. <https://doi.org/10.1080/13658816.2017.1400548>.
 47. Yang D, Cai B, Yuan Y. An improved map-matching algorithm used in vehicle navigation system. In : Proceedings of the 2003 IEEE International Conference on Intelligent Transportation Systems. 2003 IEEE International Conference on Intelligent Transportation Systems. Shanghai, China, 12–15, IEEE, 2003, pp. 1246–50.
 48. Yazdizadeh A, Patterson Z, Farooq B. Semi-supervised GANs to infer travel modes in GPS trajectories. J Big Data Anal Transp. 2021;3(3):201–11. <https://doi.org/10.1007/s42421-021-00047-y>.
 49. Zeng W, Church RL. Finding shortest paths on real road networks: the case for A*. Int J Geogr Inf Sci. 2009;23(4):531–43. <https://doi.org/10.1080/13658810801949850>.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.